Expert Systems with Applications 39 (2012) 4813–4834

Contents lists available at SciVerse ScienceDirect

Expert Systems with Applications

journal homepage: www.elsevier.com/locate/eswa

A framework for building web mining applications in the world of blogs: A case study in product sentiment analysis

Evandro Costa, Rafael Ferreira, Patrick Brito, Ig Ibert Bittencourt *, Olavo Holanda, Aydano Machado, Tarsis Marinho

Computation Institute, Federal University of Alagoas, GrOW – Grupo de Otimização da Web, Postal 15.064, 57072-970 Maceió, AL, Brazil

ARTICLE INFO

Keywords:
Semantic web
Social web
Blogs
Sentiment analysis

ABSTRACT

Recently there has been much interest in electronic commerce applications that use data mining techniques to explore datasets in the social media context. However, most of the applications have already been developed in an ad hoc manner, mainly, due to the lack of adequate tools, yielding difficulties in customizing applications and requesting high time consuming for constructing and maintaining these applications. This work addresses these problems and proposes a software framework for building Web mining applications in the blog world. The architecture of the proposed framework combines the use of blog crawling and data mining algorithms, in order to provide a complete and flexible solution for building general-purpose Web mining applications. The framework flexibility allows some important customizations, such as the construction of adapters for reading text from different blogs, and the use of different pre-processing techniques and data mining algorithms. In order to improve the efficacy of information extraction from blogs, ontology is used in the blog's description. For this, there are software agents responsible for tracking and indexing blogs related to a specific tag and for mining blog datasets. Moreover, web services are used for encapsulating existing tools and maximize reuse.

This framework has been instantiated in order to be applied for helping the blog users to effectively find out relevant information in the blog world. The focus of this paper is on describing the novel software architecture of the general framework (blog crawling and data mining) providing detailed information about the data mining sub-framework, which uses the semantic web services technology for automating service composition and consists on the main research contribution. A case study of an e-commerce application for analyzing the user's sentiment regarding specific products is reported and its results considers the effort reduction when creating a web mining application by using the proposed integrated frameworks and existing data mining tools, as well as a qualitative analysis related to quality aspects of the developed application, such as the evolution impact.

1. Introduction

Currently, the research on electronic commerce has gained particular attention to the advent of the so-called social media, which provides a rich data environment concerning user sentiment for enterprises to exploit. In this context, there has been much interest in applications that make intensive use of analysis tools present in data mining, text mining, and web mining techniques to explore datasets from social media, for example, focusing on blog datasets to blog mining applications. Additionally to the challenges on how to find out meaningful information on this vast amount of data, there are also many technical problems with respects to software engineering aspects that are involved in the development process of these kinds of applications. For instance, the majority of the existing solutions for such applications have already been developed in an ad hoc manner, mainly due to the lack of adequate tools, yielding difficulties in customizing applications and requesting high time consuming for constructing and maintaining these applications up-to-date.

This work addresses these problems and proposes a novel software framework for building Web mining applications in the blog world (Blogosphere). The architecture of the proposed framework combines the use of blog crawling and data mining algorithms, in order to provide a complete and flexible solution for building general-purpose Web mining applications. The framework flexibility allows some important customizations, such as the construction of adapters for reading text from different blogs, and the use of different pre-processing techniques and data mining algorithms. In order to improve the efficacy of information extraction from blogs, ontology is used in the blog's description. For this, there are software agents responsible for tracking and indexing blogs

* Corresponding author. Tel.: +55 (82) 3214 1401.
E-mail addresses: ebc.academico@gmail.com (E. Costa), rflm@cin.com (R. Ferreira), patrick@ic.ufal.br (P. Brito), ig.bittencourt@acm.org (I.I. Bittencourt), olavohcn@gmail.com (O. Holanda), tarsis.ms@ gmail.com (T. Marinho).

0957-4174/$ - see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.eswa.2011.09.135

related to a specific tag and for mining blog datasets. This framework has been instantiated in order to be applied for helping the blog users to effectively find out relevant information in the blog world. Moreover, the use of data mining techniques also allows complex text analysis, such as the analysis of user's sentiment based on its posts.

The software architecture of the integrated solution shows how blog crawling and data mining sub-frameworks are combined together in order to provide a valuable solution for web crawling in social media. However, since crawling is more an engineering than a research issue, the focus of this paper is on describing the software architecture of the general framework (blog crawling and data mining) providing detailed information about the data mining sub-framework, which uses the semantic web services (SWS) technology and consists on the main research contribution. The use of SWS is motivated by some limitations of the service technology. Despite the advantages of adopting a service-oriented approach, there are variabilities which are difficult to deal with in an automatic way. For example, variability related to the business process is not easy to resolve, specially when using existing solutions like BPEL (Juric, 2006). In the SWS approach used by the proposed solution, ontology is used for describing the service interface, instead of the traditional WSDL contract. It improves the framework variability and its capability of automatically identifying "compatible" services and not only services that implement a specific interface, which is possible via ontology mapping. Moreover, this feature allows the development of more autonomous applications and reduces the coupling between services.

Regarding blog crawling, the contribution consists on using ontologies for annotating the blogs before crawling; but it is not considered a master contribution of the proposed approach, since the semantic annotation of blogs is conducted manually. The adoption of software agents and services also benefits the blog crawling sub-framework, since it allows abstracting away from existing tools, such as WordPress.[1]

To evaluate this approach, the proposed framework was applied in a case study of an e-commerce application for analyzing the user's sentiment regarding specific products, based on the marking system of Technorati together with data mining algorithms. The evaluation criteria were followed in the experiment, calculating the precision, recall and *f*-measure of each test case. The obtained results considers the effort reduction when creating a web mining application by using the proposed integrated frameworks, as well as a qualitative analysis related to quality aspects of the developed application, such as the evolution impact. Although the results are considered initial, they show that our proposal is effective and relevant, specially when reusing existing tools by wrapping them as services, and when automating variability at the business process by using semantic contract descriptions, instead of WSDL. Additionally, an application for product sentiment analysis was developed to complement the case study, in order to show the advantages of the proposed framework compared to existing approaches. This application was developed by using an appropriate classifier algorithm for learning to find out the acceptance of a product in blogs.

The rest of the paper is organized as follows. Section 2 describes aspects related to Blogosphere, social media and software architecture. Section 3 introduces the proposed framework, its architecture and its implementation aspects. Section 4 present details related to the data mining sub-framework. Section 5 contextualizes the proposed framework against some related work, making the contribution of the proposed approach more explicit. A case study that validates the proposed approach as well as the obtained results are presented in Section 6. Finally, some conclusions and discussion of possible future work are presented in Section 7.

2. Background

This section describes the necessary background to understand the proposed framework and also the case study. The follow subsections describe the Blogosphere and the social media, software engineering aspects as well as the used technologies.

2.1. Social media and Blogosphere

Social media represents a broad change on how people interact with one another, allowing them easily participate in and contribute (Cho & Tomkins, 2007). Interesting tools available for interaction in social media include blogs, forums, and wikis. This work is concentrating on the blog world mining.

Blogosphere is the name given to the whole community of blogs on the Internet, it is part of the infrastructure which the ideas are developed and transmitted. Blogs are essentially only the published text of the thoughts of some author, while the Blogosphere is a social phenomenon.

According to Wikipedia the blog (a contraction of the term "web log") is a type of website whose structure allows quick update and it is usually maintained by an individual with regular entries of commentary, descriptions of events, or other material such as graphics or video. Entries are commonly displayed in reverse-chronological order, focusing on the proposed theme of the blog, it can be written by a variable number of people, according to the policy of the blog.

Technorati, one of the biggest aggregations of blogs on the Web, annually makes a survey on the state of the Blogosphere and its trends. The research of 2008 and 2009 will be used to define interesting points about the current state of blogs. In 2008, the Technorati research concluded interesting positions about the size of the Blogosphere. All studies show that blogs are a global phenomenon that hit the Web. With Fig. 1 is possible to realize interesting facts as the large number of posts that are written every day, nine thousand, and there are over 133 million blogs indexed by Technorati between years 2002 and 2008. According to the same estimate the activity in blogs doubles every two hundred days.

Two interesting questions were answered in the 2009 survey: (i) Who are the bloggers?, and (ii) The what and why of blogging. They will be answered in the sub-sections below.

2.1.1. Who are the bloggers?

The word bloggers refers to a person who write on blogs, according to Technorati research, the bloggers in general, are part

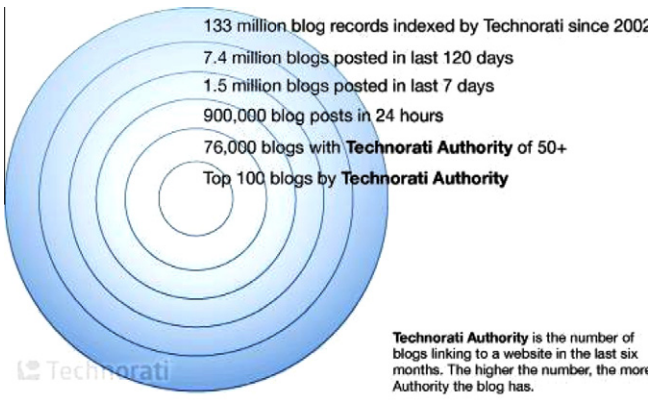

Fig. 1. A quick journey into the size of the Blogosphere.

[1] wordpress.org.

of a group of highly educated and rich. Almost half of all bloggers surveyed have a graduate degree, and most have a family income of $75,000 or more per year. Besides these data, other interesting features are:

- 60% are 18–44.
- The majority are more affluent and educated than the general population.
- 75% have college degrees.
- 40% have graduate degrees.
- One in three has an annual household income of $75 K+.
- One in four has an annual household income of $100 K+.
- Professional and self-employed bloggers are more affluent: nearly half has an annual household income of $75,000 and one third topped the $100,000 level.

These features indicate that people who are using blogs are qualified, then the published texts provide interesting information.

2.1.2. The What and Why of Blogging

Self-expression and sharing expertise continue to be the primary motivations for bloggers, and 70% of all respondents say that personal satisfaction is a way they measure the success of their blog. Among the professional bloggers, however, the leading metric of success is the number of unique visitors. Hobbyists bloggers overwhelmingly blog about personal musings while professional and aspiring professional bloggers tend to be more topical.

The rise of the professional blogger continues, where 70% of part-timers and self-employeds are blogging more than ever, while Hobbyists are blogging somewhat less. The key driver of decreased blogging is an increase of work and family commitments (64%). 30% of those who are blogging less say it is because they are devoting more time to microblogging and social networks. Bloggers describe significant, positive impacts on their personal lives, but even more bloggers have experienced positive career and business impacts. 70% say that they are better known in their industry because of their blog. The diversity of the Blogosphere, Fig. 2, and the passion for sometimes very niche topics, is also reflected in this question even given 23 choices including most broad fields of inquiry, 30% of respondents say that their primary subject is other.

2.2. WebCrawlers and mining

The search engines are becoming increasingly essential to obtain relevant information from large amount of data that a Web offers. Search engines store huge collections of pages with the help of web crawlers, which are responsible for traveling by Web through links and to collect data that are usually indexed to the user could runs queries efficiently (Gupta & Johari, 2009).

According to Gupta and Johari (2009), Web crawler is a computer program that exploits the graph structure of the Web to move from page to page. As already mentioned, the greatest motivation to create Web crawlers was to recover the Web pages and add them or their representation in a local repository to facilitate the search for the user. In its simplest form a crawler starts of a page and from there use external links to reach the other pages.

If the Web is a static collection of pages, then come a time when the crawlers were no longer needed, since all the pages you would be linked to a repository. However, the Web is a dynamic entity that evolves with different rates and speeds. Hence there is a continual need for crawlers to help applications stay current as new pages are added and old ones are deleted, moved or modified.

Below, we describe details of the flow activities and the basic architecture of a Web crawler.

2.2.1. Flow activities

Fig. 3 shows the flow of a basic sequential crawler, it will be detailed below:

- *Initialize frontier with seed URLs:* The frontier is the to-do list of a crawler that contains the URLs of unvisited pages. For this step can be implemented, for example, a FIFO (First in, First Out) scheduling scheme to sort the URIs that will be accessed.

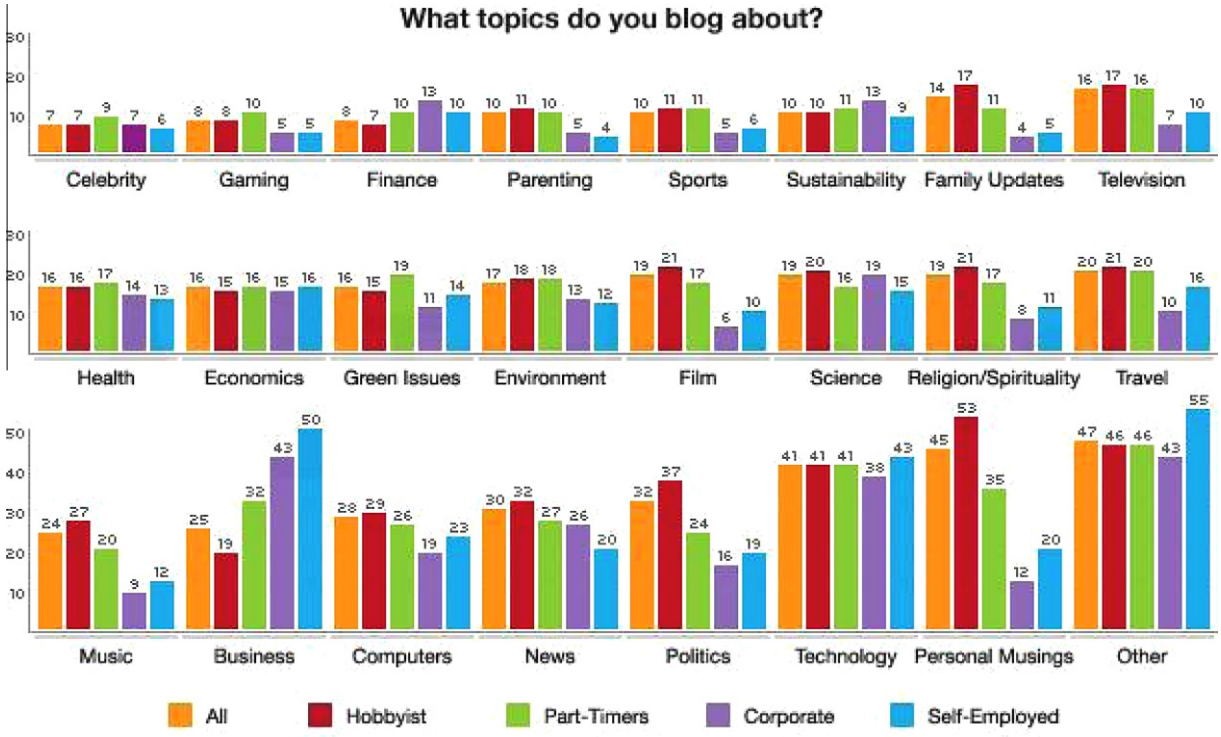

Fig. 2. Blogged topics.

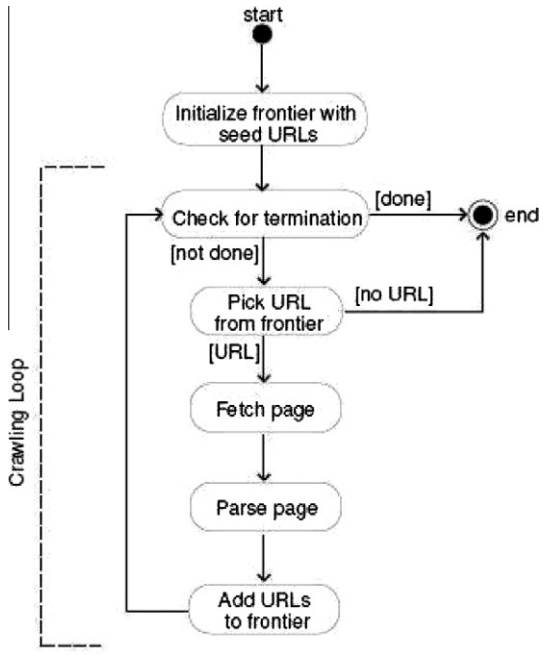

Fig. 3. Flow of a basic sequential crawler.

- *Check for termination:* Checks if the frontier is empty, if not returns the next URI to be analyzed.
- *Fetch page:* Obtain the page, typically through a request made through an HTTP client. Deal with connection problems and determines some characteristics of the pages, as the last update.
- *Parse page:* Creates a parser on the page to extract useful information and possibly guide the next step of the crawler, moreover, seeks to eliminate information not expressive or useless.
- *Add URLs to frontier:* Adds URIs, obtained in the previous step, to the frontier.

2.2.2. Basic architecture

Fig. 4 shows the basic components of crawler architecture.

In this architecture there are two fundamental components that are the application and system crawlers. The system crawler has a responsibility to go to the Web and download pages. Moreover, the application one is responsible for passing a list of pages that must be downloaded and make appropriate operations with the data obtained by the system crawler.

2.2.3. Social Crawler and mining

Every day is greater the use of social tools such as wikis, emails, forums, blogs, tools, relationship, among others. These tools contain information that can be valuable if used properly, for example, a networking tool describes some characteristics of users that can be used to profile him and mechanisms for personalization services.

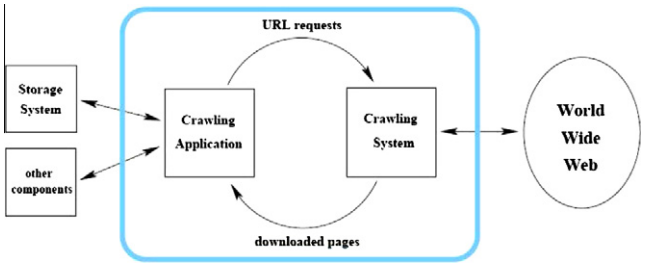

Fig. 4. Basic two components of the crawler.

However, this requires mechanisms to obtain information from these tools. Then coming into the social crawlers (Hurst & Maykov, 2009), they are similar to crawlers previously described, but have a fundamental difference, they are designed to gather specific information from tools and not the entire page. For example, a Blog Crawler is a mechanism that obtains information from blog posts, not the entire web page of the blog.

Data mining algorithms have been applied to explore social media, especially in the context of blog mining applications in electronic commerce. To this end, classifiers such as Naive Bayes and decision tree as well as clustering tools such as k-means and neural networks, particularly self-organizing maps (Witten & Frank, 2005) have been used.

2.3. Used technologies

Some technologies were used in this paper (e.g. Lucene, Weka) are explained as follows.

Lucene includes a library of reusable code developed by the Apache organization. Among other characteristics, Lucene provides high performance on indexing, efficient search algorithm and it can perform several tasks such as, for example:

- Ranking Search: the best results appear first.
- Many query types: queries phrases, queries with advanced lexical operators, queries by proximity.
- Search by attributes (title, author, content).
- Search by data range.
- Sort by field.
- Simultaneous updates and searches.

The data source, its format or the language that it was written does not matter for Lucene, since that these data can be converted to text. This means that Lucene can be used to index and search stored data in: files, web pages in remote servers, stored documents in the system of local files, text files, Microsoft Word documents, HTML documents, PDF files or any other format which can be extracted text information. Lucene is developed in Java, there are several applications that run using it, it is open source and it has a large documentation in various languages, including a book, (Hatcher & Gospodnetic, 2004), which details several aspects of the tool.

Weka is a collection of machine learning algorithms intended for common problems in data mining like regression, classification, association rules and attribute selection (Witten & Frank, 2005). It also includes tools for data preprocessing, learning models creation, results analysis and classifiers performance. All Weka functionalities are accessible through a simple and intuitive graphical interface. Besides, Weka provides an API that allows using its algorithms and tools directly in Java code, which enables the creation of tools. In addition, the proposed framework could also use tools like RapidMiner and Natural Language Toolkit.

2.4. Software architecture

The *software architecture* of a program or computing system is the structure or structures of the system, which comprises software elements, the externally visible properties of those elements (architectural components and connectors), and the relationships among them (architectural configuration) (Bass, Clements, & Kazman, 2003). Since the software architecture represents the structure of the software system, it is a set of significant decisions about the software organization. Those decisions concerns the selection of the structural elements and their interfaces by which the system is composed, together with their behaviour as specified in the collaborations among those elements (Kruchten, Henk Obbink, & Stafford, 2006). Moreover, since software architecture deals

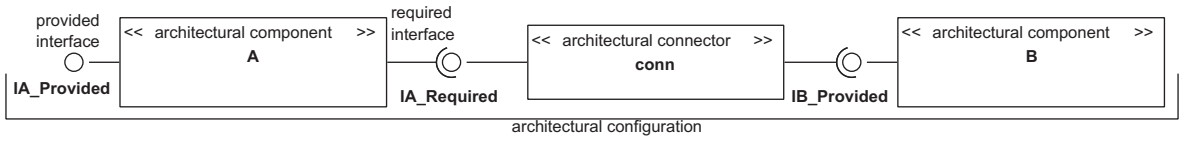

Fig. 5. Example of an architectural configuration in UML.

with the design and implementation of the high-level structure of the software, its decisions have a decisive impact into the quality attributes of the whole system, such as availability, reliability, and testability.

Fig. 5 presents an example of an architectural configuration involving two components and a connector. *Architectural components* (e.g., A and B) are primary computational elements or data stores of a system and are defined through the list of operations that they provide (provided interfaces), as well as the operations necessary for executing their functionalities (required interfaces). To exemplify the concept of provided and required interfaces, component A of Fig. 5 has a provided interface called IA_Provided, and a required interface called IA_Required. *Architectural connectors* (e.g., conn) are mediators of the communication and coordination activities among components. That is, they define the rules governing component interaction and specify any auxiliary implementation mechanism required. The way that components and connectors are connected together is defined by the *architectural configuration*.

Besides representing the structure of a software system, the software architecture can also represent its behaviour through architectural scenarios, which are relevant at the architectural level because they enhance the structural representation of the system with an abstract representation of its behaviour.

2.4.1. Architectural principles

Create, maintain and evolve the architecture of a system is a difficult task, which is usually guided by the design decisions of the architect and the development team. Due to the subjective nature of these design decisions, the development team and especially the software architect must exercise their abilities to recognize the best solutions. To assist in this task, the software architect can be based on a list of principles that should always be pursued. We following present the six basic principles of software architecture:

1. *Encapsulation.* The architecture is organized into relatively independent parts, in order to be an abstract representation of a software solution, in which the implementation details are omitted.
2. *Low coupling.* Due to the encapsulation already discussed, the communication between the architectural components must occur through well-defined communication protocols.
3. *High granularity.* One of the main advantages of specifying the software architecture is the ease for representing complex structures. The comprehensive understanding of the system is a result of abstraction afforded by the architecture. Thus, the components that compose an architecture typically represent a role or category, which may be detailed through the definition of an internal architecture.
4. *High cohesion.* Cohesion represents the degree of closeness between groups of features. The high cohesion of architectural components is a consequence of the definition of clear and easily understood roles.

2.5. System quality attributes realized by the software architecture

In conferences on software architectures, often refers to how properties/quality attributes of the system are satisfied by the architecture, in contrast, the functional properties of the architecture are assumed implicitly. A functional property deals with some particular aspect of system functionality and is usually related to a functional requirement. In the past, software developers have focused mainly on functional properties of the systems. However, in modern software systems, non-functional properties are becoming increasingly important. A non-functional requirement, also known as a quality attribute, denotes a characteristic of a system that is not covered by its functional description. A non-functional property is typically related to aspects such as reliability, adaptability, interoperability, usability, persistence, maintainability, security and distribution. The non-functional properties of software architectures have a major impact on the software development, maintenance and extension. The larger and more complex software system is and the longer its lifetime, the greater the importance of their non-functional properties.

As a result, the design decisions that should be taken during the architecture design are often complex and involve many aspects of the system's non-functional properties. Aiming to make good design decisions, it is essential to well define the context of the problem, so that you can identify the best choices among the various design options available. A very useful way to clarify this context is to apply the separation of concerns (Parnas, 1972). In order to separate the various aspects involved in software architecture design, it is necessary to establish the conceptual boundaries between the architectural elements. Each partition can be responsible for solving a subset of the expected requirements. In this context, the software architect is responsible for composing the different partitions so that all requirements are satisfied.

2.5.1. Architectural styles and reference architectures

An architectural style defines a set of types of components and connectors which can be used to compose a system and a set of restrictions on the possible architectural configurations that can be created (Mary & Clements, 1997).

The most popular architectural styles were initially described in an informal way and not systematized through discursive definitions and illustrative diagrams. The pattern community has dedicated effort for documenting proven solutions, including architectural styles in the context of specific types of problems in which each solution applies (Mary & Clements, 1997). These "applied architectural styles" are known as architectural patterns (Buschmann, Meunier, Sommerlad, & Stal, 1996). An architectural style defines a family of software architectures at a higher level of abstraction than a specific configuration of components and connectors. The analysis of an architectural style in this high-level of abstraction allows us to deduce properties that will exist in any configuration that can be created in accordance with that style. These are general properties of architectural styles which motivates the choice of one or more architectural styles to guide the architectural design in particular applications.

A reference architecture is a proven template solution for an architecture of a specific domain. This concept is used by important software development processes, such as the Unified Process (Jacobson, Booch, & Rumbaugh, 1999), and it is considered a resource containing a consistent set of architectural good practices for guiding all the development phases. Architectural styles can be seen as

reference architectures, when they are known as successful solutions for specific domains. Reference architectures are specially important since they show how to compose the parts of a solution together, in order to produce a specific result.

2.6. The layered architectural style

In software development it is necessary to partition the total system into cohesive and loosely coupled subsystems (Larman, 1998). Subsystems are usually identified by the services they offer, which are defined through the operations of its interfaces. Consequently, one of the most important activities of the architectural design is the specification subsystems' interfaces. Furthermore, a subsystem may both depend on other subsystems when assume the role of client, and offer services that are used by other subsystems when assume the role of server.

Pressman (2001) shows a list of design criteria with which subsystems in any object-oriented development project must comply:

- The subsystem must have a well defined interface through which all communication occurs with the rest of the system.
- With the exception of a small number of "communication classes", classes in a subsystem should collaborate only with other classes within the same subsystem.
- The number of subsystems should be small, where the meaning of "small" depends on the project and should be judged by the software architect, who normally is an experienced member of the development team.
- A subsystem can (and should) be partitioned internally to reduce its complexity.

When a system is partitioned into subsystems, another activity occurs concurrently: the division in layers. Each layer of a system contains one or more subsystems and is responsible for a set of related functionality. Typically, in systems structured in layers, subsystems located in a particular layer n only request services offered by sub-layer $n+1$ (immediately below) and only offer services to the subsystems of the layer $n-1$(immediately above). Besides, the layers $n-1$ and $n+1$ not "see" each other directly (Bass et al., 2003). Thus, as can be seen in Fig. 7, a layer of its top layers hide implementation details of the lower layers, facilitating the update of a specific layer without change the entire system.

A layered structure which is widely used for constructing information systems, especially web-based applications, is based on three layers: (i) view layer, which is responsible for the interaction between the system and its user; (ii) control layer, which is responsible for implementing the business rules related to the application;

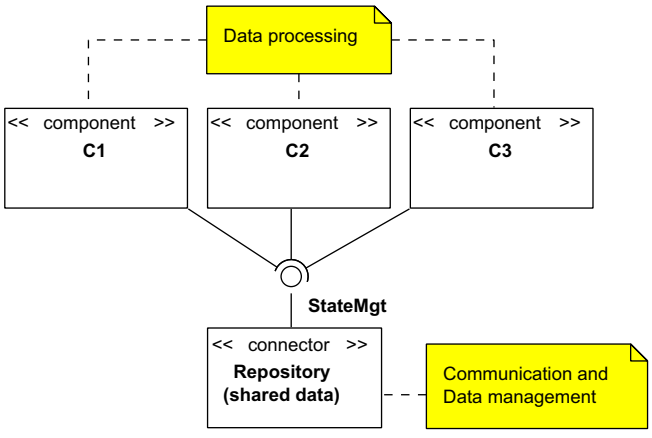

Fig. 6. Example of the repository architectural style.

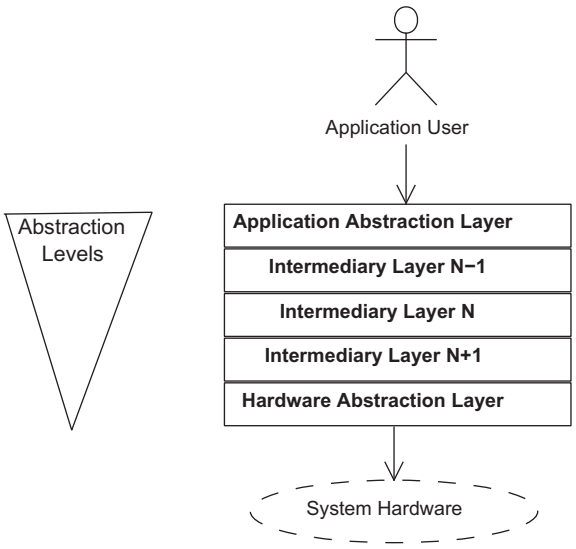

Fig. 7. Example of a layered architectural style.

and (iii) model layer, which handles the storage of data. Software architectures organized under this division are known as three-tier architectures or "model-view-controller (MVC) architecture".

2.7. The repository architectural style

The repository architectural style is characterized by the presence of two types of components: (1) the repository itself, a structure that stores and centralizes the system data; and (2) a set of independent components, which use and update the repository data. The communication between these systems occurs through data sharing, since they use the same repository (Shaw & Garlan, 1996). Fig. 6 illustrates an architecture following the repository style. There, the repository represents and shares the general state of the system, while components (C1, C2 and C3) performs processing, communicating directly with the repository.

Examples of systems that typically use this style include computer aided design (CAD) tools and integrated development environments (IDEs), which usually integrate several modeling tools, visualization, support for editing, and other features based on a single same data set.

Regarding the control flow of this architectural style, it is possible to have two variations: (1) when an input system triggers the processes to be executed; and (2) when the internal state of the repository triggers the processes to be executed. The latter case illustrates the typical operation of blackboard architectures.

2.8. The pipes and filters architectural style

According to the pipes-and-filters architectural style, the system components are executed in series, according to a set of consecutive inputs and outputs (Shaw & Garlan, 1996). Each component reads streams of data on its inputs and produces streams of data on its outputs. Local transformations are performed on the input streams incrementally. In this style, components are called filters, since they are responsible for transforming data; and the connectors are called pipes, since they conduct the output of a filter as the input of another.

An important feature of this style is the independence of filters, since they should not share their state each other. Another important feature is the fact that a filter does not know other filters, even the ones it interacts with. The filters' specifications should be

restricted to their input and output pipes, and should not identify the components at the end of each pipe. Fig. 8 illustrates this style.

The most famous example of a pipes-and-filters architecture is the Unix shell. Unix uses this style to connect its components (represented by Unix processes). Other examples of pipes-and-filters applications are compilers, some signal processing systems, parallel programming, functional programming and distributed systems.

Furthermore, systems that follow this approach may impose rules of communication in their pipes in order to standardize the communication and facilitate the modification of their filters. These rules in their pipes can be a disadvantage, since they may require an extra work of the filters for handling their input and output streams in order to follow the rules of communication. It can affect the performance and construction of the filters. A possible solution for this problem can be the use of adapter pipes, thus removing the responsibility of handling data conformance from the filters.

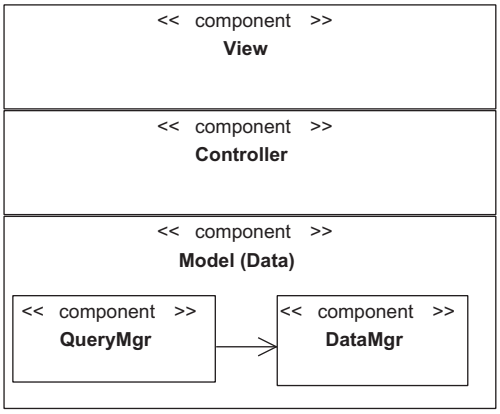

Fig. 9. Example of a heterogeneous architectural style.

2.9. Heterogeneous architectural styles

Sections (2.6)–(2.8) have presented some of the main "basic" architectural styles which are cataloged in the literature. Despite the importance of understanding the individual nature of each style, the implementation of real systems almost always involves a combination of many of them.

Architectural styles can be combined in various ways. The most common way of combining styles is through hierarchies. A component of a system that is already inserted into any architectural style can have an internal structure itself, which can be a totally different architectural style of the system to which it belongs. Fig. 9 presents a system of three layers. However, the model layer uses the pipes-and filters architectural style internally.

What is more surprising is the fact that connectors can also be hierarchically decomposed. For example, a pipe connector can be implemented internally as a "First In, First Out" (FIFO) queue, which provides operations for inserting and removing items.

A second way of combining architectural styles is to allow a simple component using various types of connectors. In other words, it is possible that a particular component needs to communicate with other components in different ways and using different protocols.

2.10. Semantic web services

The semantic web services are based on creation of semantic descriptions to traditional Web Services through ontologies in order to provide an scenario where intelligent agents become capable of exploring the semantic descriptions of those services with the purpose of performing complex tasks without the direct interference of humans (Cho & Tomkins, 2007). In this manner, Fig. 10 presents the semantic web technologies.

As it can be observed from Fig. 10, the semantic web services present features of the traditional *Web Services*, but also the ability to automate current processes of the *Semantic Web*. The Fig. 11 illustrates exactly this idea, where semantic web services are the union of these two technologies (Daconta, Obrst, & Smith, 2003). The automation process on the semantic web occurs during discovery, composition and invocation of services, allowing the execution of these processes without direct interference from Human Agents. Since this process occurs without human interaction, it is possible to provide mechanisms to monitor and evaluate the execution of these processes automatically in order to improve results.

In this context, the use of semantic web services allows the building of dynamic systems based on a set of services executed to provide some system functionalities only during the execution time and without the need of interference from Human Agents. This is possible only because the processes of discovery and invocation of services are automated (McIlraith, Son, & Zeng, 2001).

Thanks to the flexibility of the semantic web services, systems based on this technology can provide mechanisms to ensure its autonomy in both the execution of their functional requirements and the non-functional ones. For example, considering a problem in the service execution due to a lack of one specific service, the system can automatically select in its repository a service that is similar to the missing one and executes it transparently to the user. This is possible because each service provides its semantic descriptions and therefore an algorithm can match service description and

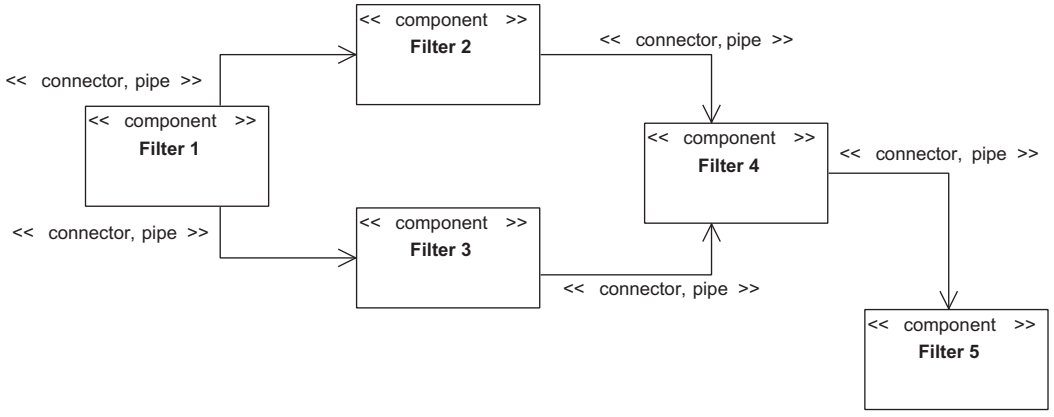

Fig. 8. Example of the pipes-and-filters architectural style.

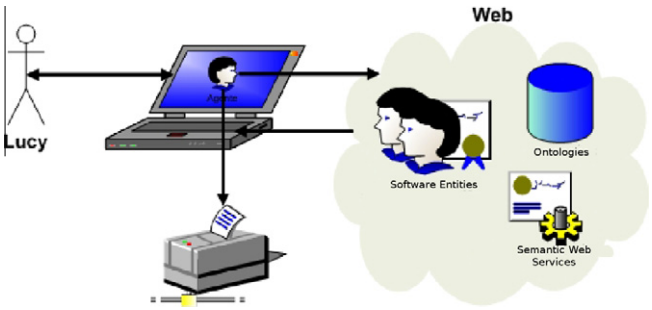

Fig. 10. Semantic web technologies.

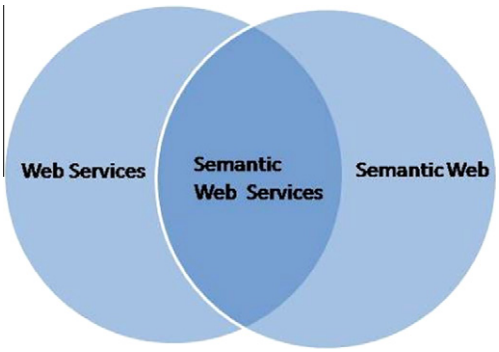

Fig. 11. Semantic web services.

thus ensuring the functioning of the system before errors due to missing services.

In the core of any work based on SWS are the semantic descriptions or annotations of services that support automation tasks such as their discovery, selection and composition. Therefore, there are four member submissions for semantic annotations fo web services: WSDL-S (WSDL-S, 2010), SWSF (Boley, Grosof, Battle, Bernstein, & Gruninger, 2005), WSMO (Travers, Domingue, Polleres, Bussler, & Burdett, 2005) and OWL-S (Hobbs et al., 2004). Nevertheless, the OWL-S approach is presented above.

2.10.1. OWL-S

OWL-S (Hobbs et al., 2004), formerly known as DAML-S, is a set of ontology used for describing semantic web services expressed in the Web Ontology Language (OWL). OWL-S was submitted to W3C in 2004 and it defines an upper ontology for semantically describing Web services along three main aspects:

- The Service Profile describes "what the service does" in terms of inputs, outputs, preconditions and effects (IOPEs).
- The Service Model describes "how a service works" in terms of a process model that may describe a complex behaviour over underlying services.
- The Service Grounding describes "how the service can be invoked", usually by processing an WSDL file.

Fig. 13 depicts the ontologies that compose the OWL-S. In summary, the building of descriptions for services based on OWL-S consists of creating instances of ontologies proposed in these approaches.

3. Architectural design of the proposed integrated framework

For designing the software architecture of the proposed integrated framework, we have followed two complementary activities: (1) choose a reference architecture; (2) refine the reference architecture with other architectural styles. In order to choose a good reference architecture, we have analyzed the characteristics of the framework domain. In general, the framework should provide two high-level functionalities: (1) text extraction, which consists on Web crawling and (2) data mining, which consists on extracting information from Web content.

Since the two aforementioned high-level functionalities should be executed in sequence, and the provided output of the text extraction is the required input of the data mining, the reference architecture chosen for the framework follows a heterogeneous architectural style (Section 2.9) based on the pipes-and-filters architectural style, presented in Section 2.8. So, as presented in Fig. 12, the abstract view of the software architecture consists of two complementary components (filters), and a connector (pipe) between them. Component Text Extractor is responsible for extracting textual content from Web content, specially blogs in the context of this article; while component Information Extractor is responsible for extracting extra information from the textual content received from the Text Extractor through the Text Adapter connector. The component WebMiningController is responsible for initiating the process of extracting information from blogs, which is done through the IBlogInfo interface of the TextAdapter connector. Interface IWebMining is provided for applications which use the framework for extracting information from Web content. In order to better explain the integration between the two parts of the framework, Fig. 14 presents a sequence diagram involving the orchestration involving Text Extractor and Information Extractor components.

Regarding the processing of text before executing data mine algorithms (preprocessing), the proposed solution presents two levels of preprocessing: (1) HTML-level preprocessing, which is done by the Text Extractor component after reading the content of an HTML page (e.g., remove white spaces, clean HTML tags) and (2) content-level preprocessing, which is done by the Information Extractor component, in order to make the execution of the algorithm faster (e.g., remotion of non-relevant texts in terms of a specific purpose).

As illustrated in Fig. 14, to use the proposed framework it is necessary to invoke services of IWebMining interface of component WebMiningController. This component starts the information extraction process by requesting the respective service from the IBlogInfo interface of the TextAdapter connector. Since such architectural connector implements the pipes-and-filters protocol, it first requests the extraction of texts from blogs, through IBlogCrawling interface, and then, the information extraction, giving the extracted text as input for the IInfoExtractor interface.

After identifying the reference architecture of the proposed framework, the next step was to refine such architecture in order to detail the internal structure of the two high-level components. The next Sections 3.1 and 3.2 present the internal structure of the Text Extractor and Information Extractor components, respectively.

3.1. Text extractor internal architecture

The Text Extractor component consists on a (sub) framework for extracting texts from Web content. For this, it defines components for preprocessing, indexing, extracting HTML tags, and other functionalities necessary for improving the quality of the text extraction from HTML pages. Since the Text Extractor component has internal variabilities which can be personalized, Fig. 15 presents a feature model containing the possible decisions involving its configuration. According to the notation used in this figure, an arch represents an alternative option (semantic of XOR operator) and a circle represents an optional selection (individual and indepen-

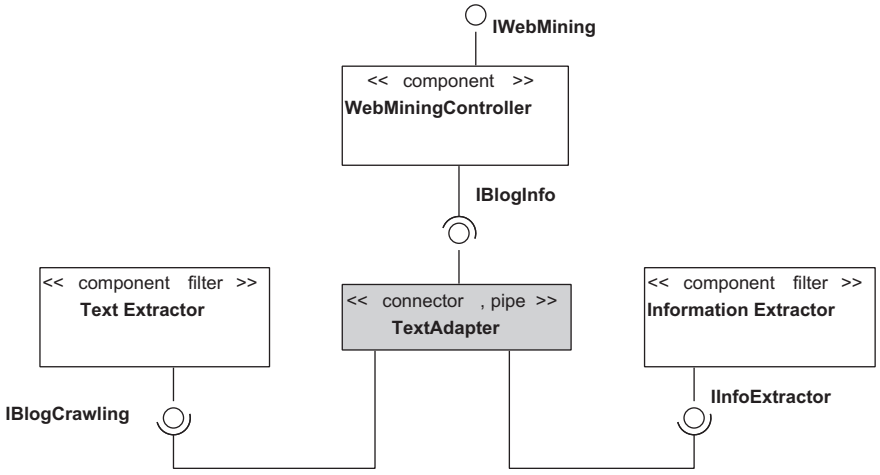

Fig. 12. High-level view of the framework's software architecture.

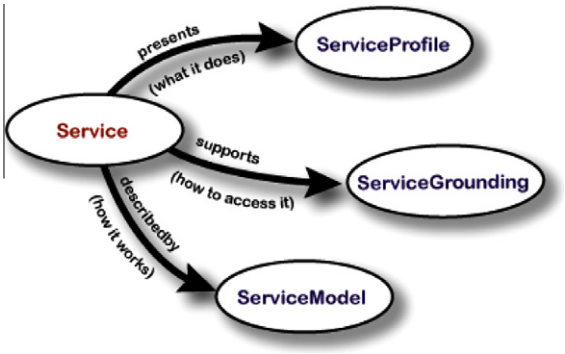

Fig. 13. OWL-S ontology.

dent decision). Further, these variabilities were implemented as hot-spots of the framework at the detailed design phase.

The internal architecture of the Text Extractor component, presented in Fig. 16, follows the layers architectural style, presented in Section 2.6. Three layers have been defined: (1) control layer, which contains components responsible implementing the basic activities for extracting text from Web content; (2) I/O layer, which contains the component responsible for reading text content; and (3) source layer, which contains the target repositories to be mined, which can be either Web-based repositories (using HTTP protocol) or database–based repositories (using ODBC protocol). The existence of repositories with different technologies facilitates the integration of the proposed framework with other existing applications which generate text content. This characteristic can be seen as an implementation of the repository architectural style (Section 2.7).

As illustrated in Fig. 17, the IBlogCrawling interface provides services of text extraction from Web content. After receiving a service request, the Controller component obtain the text content through the IGetText interface of the I/OMgr component, which can get either HTML pages (by using the HTMLAdapter) or get content in a database (by using the DatabaseAdapter) through the ISourceData interface. Then, the content of the HTML page is preprocessed by the Pre-Processor component, which is requested by the controller through the IPreProcessing interface. The preprocessing activity consists on a set of configurable activities, such as clean HTML, remove white spaces, filtering information, stemming the content, and other specific preprocessing technique implemented by the

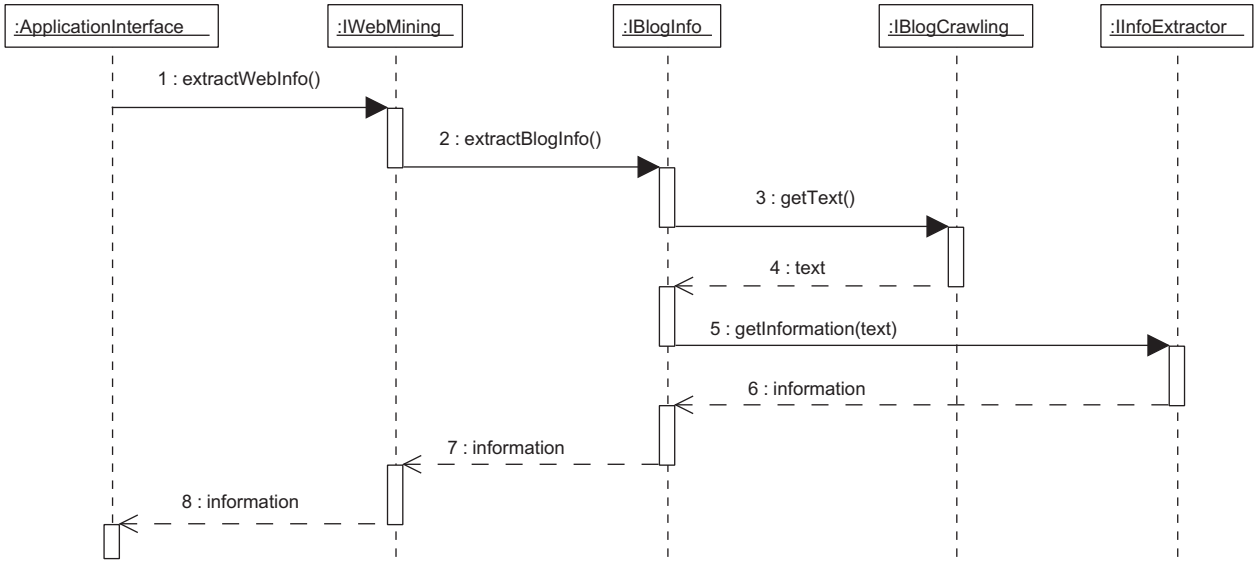

Fig. 14. Sequence diagram of the high-level software architecture.

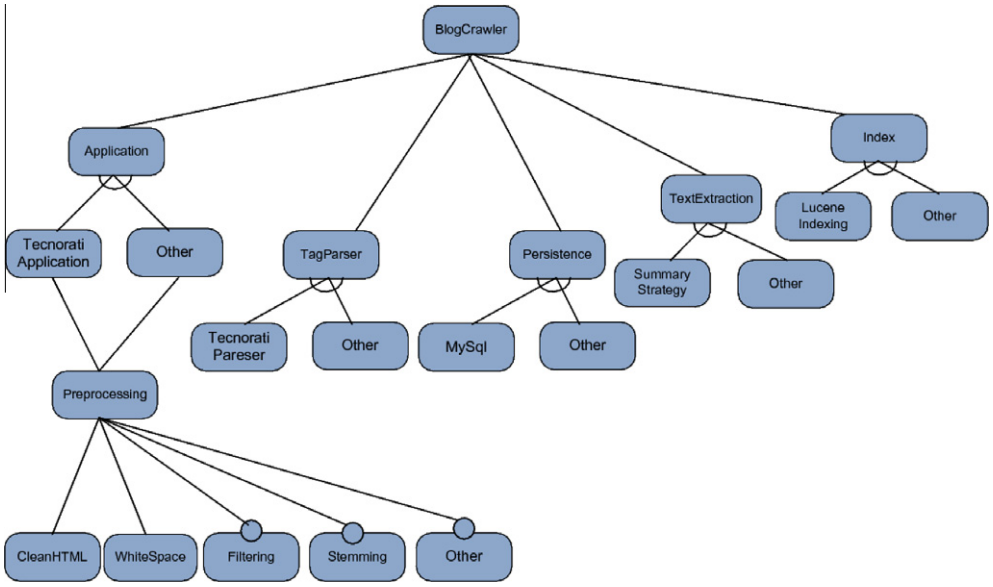

Fig. 15. Feature model of the Text Extractor component.

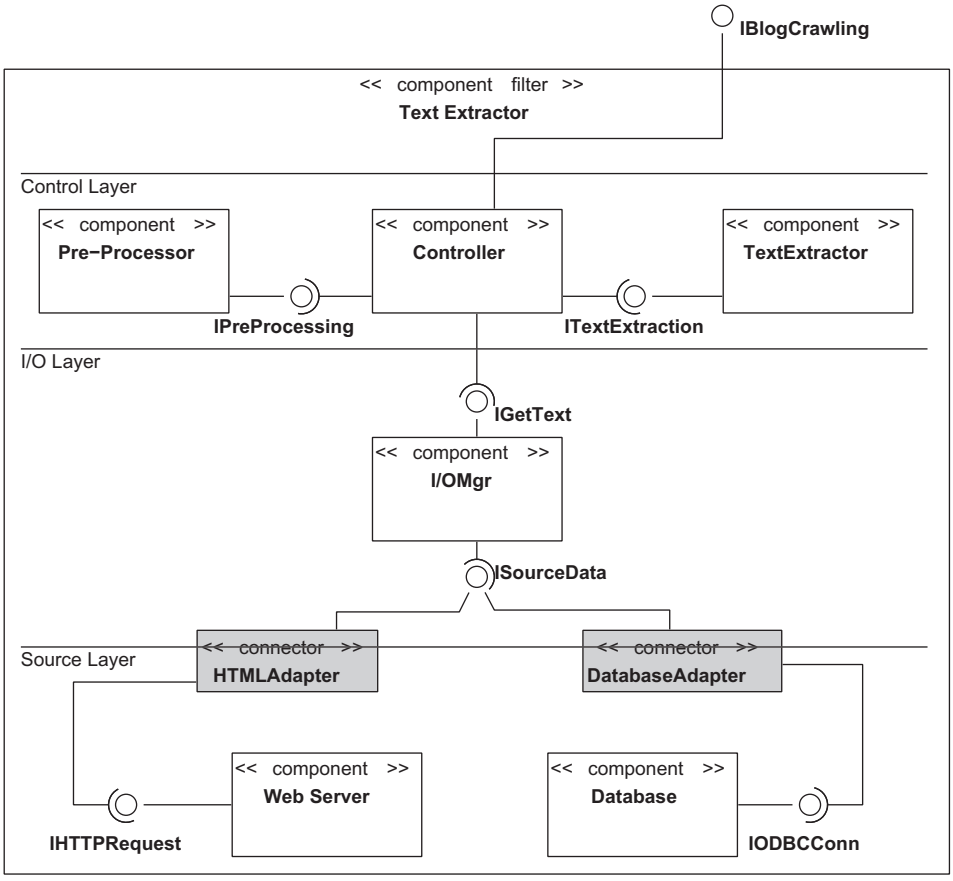

Fig. 16. Internal structure of the text extractor architectural component.

user. Finally, after the preprocessing, the text content can be extracted through ITextExtraction interface of the TextExtractor component. This extraction consists on process the textual content in order to improve the semantical quality of the text by using specific techniques, such as summarizing techniques. The user can also define new techniques when instantiating the framework.

3.2. Information extractor internal architecture

The Information Extractor component consists on a (sub) framework for extracting information from texts by using data mining. This framework follows a service-oriented approach in order to facilitate the extension and integration with different applications

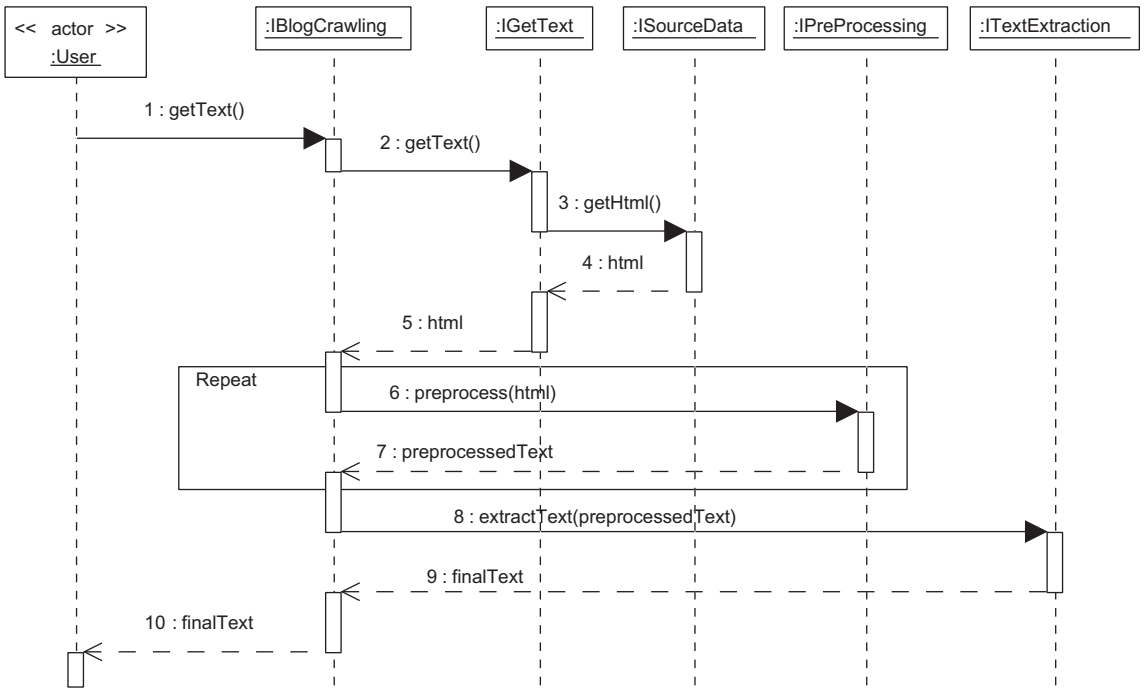

Fig. 17. Sequence diagram of the text extractor component.

over the Web. For dealing with data mining, the framework defines components for preprocessing the text, and chooses different mining algorithms, according to the purpose of the application. In the context of this article, the framework was used for developing an e-commerce application. Details about the specific application is presented in Section 6 in the context of a case study. The internal variabilities of the Information Extractor component is presented in Fig. 18, in a feature model containing the possible decisions involving its configuration.

The internal architecture of the Information Extractor component, presented in Fig. 19, follows a heterogeneous architectural style, which combines the layered architectural style with the Service-Oriented Architecture (SOA). The difference between the software architecture actually defined and the layered architectural style is the presence of a special component, called Service Identification, which is responsible for describing and locating all the services defined by the application. Besides the aforementioned special layer, other component has also been defined: the Data Mining component, which receives a text as input and is responsible for mining it in order to extract extra information, such as emotional information and feedback about the sentiment related to a specific subject. Due to it importance and complexity, the Data Mining component has an internal architecture, whose details are also presented in this section (Fig. 21).

As illustrated in Fig. 20, the information extraction starts when a request is received by the IInfoExtractor interface of the Controller component, together with the text that should be mined. This component starts the information extraction process by locating the respective data mining service which should be executed. The location occurs through the ILocateService interface of the Service Identification component. Then, a request is propagated to the respective service, which implements interface IDataMining of the Data Mining component, which is responsible for extracting information from text content.

The internal structure of the Data Mining component follows a heterogeneous architectural style which combines the pipes-and-filters architectural style (Section 2.8) and the layered architectural style (Section 2.6). The pipes-and-filters architectural style has been used for defining the data mining process, which is composed by three basic activities: preprocessing, data mining and postprocessing. These activities are implemented as Web services and structured as presented in Fig. 21. The Mining Preprocessing component contains services that prepare the text data before it can be mined. The Mining Services component contains services that encapsulate mining algorithms and tools as Web services. The Mining Postprocessing component contains services that implement postprocessing algorithms according to the specific application in which the framework is being used. In the context of this article, the Mining Postprocessing component was configured for implementing e-commerce functionalities (e.g., recommend a product based on the user sentiment about it). The orchestration among the services of preprocessing, data mining and postprocessing is done by the Service Manager component, which is a Web service responsible for discovering, composing and invoking the available semantic web services at the Data Mining component.

Besides, in order to support the execution of the data mining process, two utility components are also defined: (1) Mining Algorithms component contains algorithms that can be used for preprocessing (used by the Mining Preprocessing component) and data mining (used by the Mining Services component) and (2) Ontology component, which details the semantic descriptions of the Web services (semantic Web services) thus allowing dynamic identification and orchestration involving services.

Fig. 22 presents the behaviour of the Data Mining component, after receiving a request in the IDataMining interface (see Fig. 20). After receiving a service request and the respective text to analyze as an argument, the Service Manager component requests the execution of the data mining process through the IMiningProcess interface of the PFConnector connector. Since this connector follows the pipes-and-filters protocol, it executes the three following components in sequence, by orchestrating their services. First, a service of Mining Preprocessing is located (through ILocateService required interface) and executed (IPreprocessing). Preprocessing services can use algorithms provided by the Mining Algorithms component

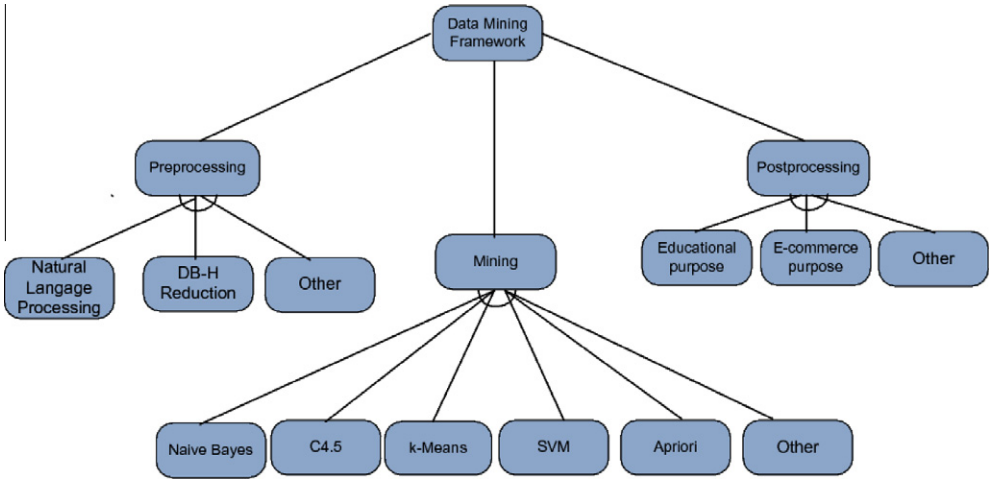

Fig. 18. Feature model of the Information Extractor component.

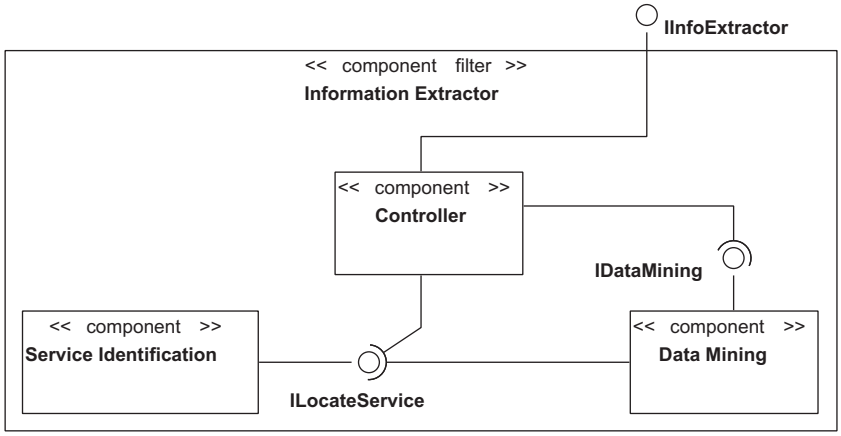

Fig. 19. Internal structure of the information extractor architectural component.

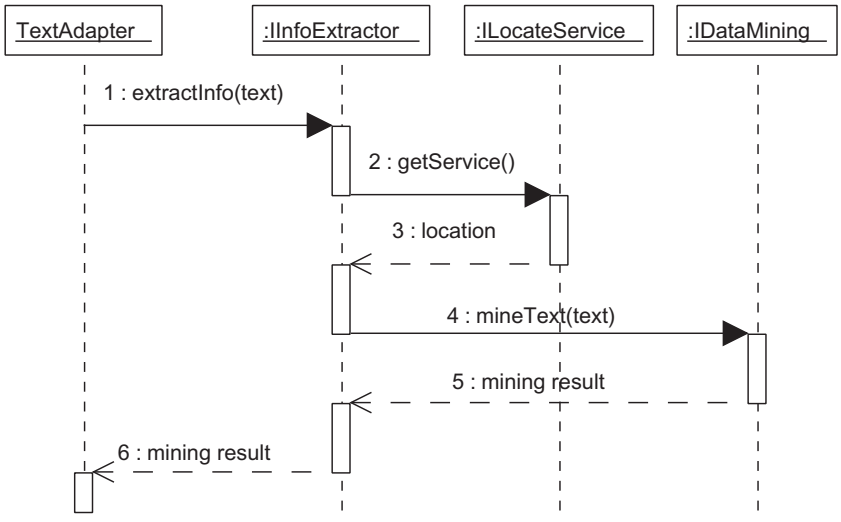

Fig. 20. Sequence diagram of the information extractor component.

(IMiningAlgorithms interface). After preprocessing the text, a mining service is located and executed passing the preprocessed text as an argument to the IMining interface, which can also use provided mining algorithms (IMiningAlgorithms interface). Finally, the mining results should be postprocessed according to its specific purpose, which in the context of this article is e-commerce. For this, the PFConnector connector locates a postprocessing service and executes it passing the data mining results as an argument

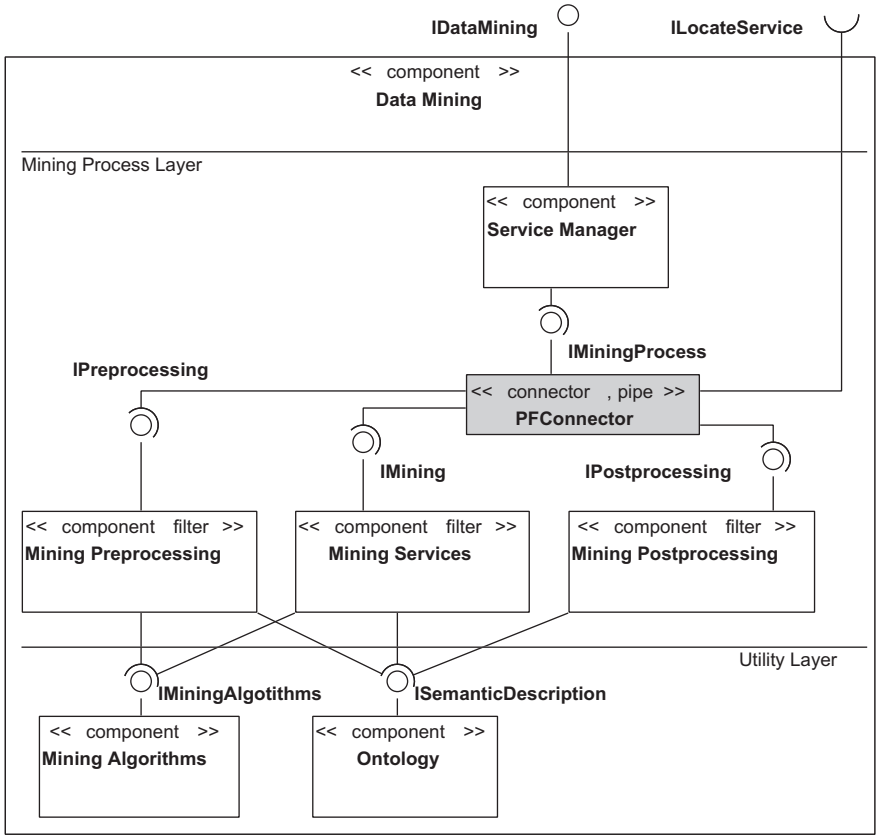

Fig. 21. Internal structure of the data mining layer.

(IPostprocessing interface). All the data mining services are semantically specified using ontology (ISemanticDescription interface).

4. Data mining framework based on semantic web services

Although the data mining solution presented in this article is a framework by itself, it is part of the whole solution that also involves blog crawling (Section 3). In order to make it explicit, we refer the data mining solution as a "sub-framework". This section presents details regarding the data mining sub-framework. First, Section 4.1 justifies the design decision of using services and more specifically semantic web services and ontologies. Then, Section 4.2 presents how information regarding the data mining sub-framework maximizes reuse and automation by using semantic web services.

4.1. Design decisions

Although traditional object-oriented frameworks provide the reuse and adaptability according to the needs of specific applications, their use is usually restricted to applications in specific programming languages. In order to promote interoperability and reuse of existing tools for pre-processing and data mining, the proposed framework adopts the service-oriented approach (Erl, 2005). So, services are used for encapsulating selection, pre-processing and data mining algorithms, as well as domain-specific functionalities to be reused across applications. Besides interoperability, the adoption of services also improves the system integrability, since it consists on invoking a Web service over the Internet, what is a simple task. Finally, the wrapping of existing tools and algorithms also facilitates large-scale reuse and improves software flexibility.

Despite the advantages of adopting a service-oriented approach, there are variabilities which are difficult to deal with in an automatic way. For example, variability related to the business process is not easy to resolve, specially when using existing solutions like BPEL (Juric, 2006). Even when considering a standard business process for data mining (selection, pre-processing, transformation, data mining, and post-processing), it is difficult to guarantee if the services will be discovered, composed and invoked in an automatic fashion. In traditional approaches such as BPEL, the developer has to predefine the services that should be used in each activity.

The adopted solution for allowing automatic discovery, composition and invocation of Web Services, was the use of complementary semantic description of services, which allows automatic interpretation by using processing algorithms. The combination of Web Services and a complementary semantic is known as semantic web services Subsection 2.10. In this approach, the flow of services does not need to be predefined, since the entire process is conducted automatically from the service discovery, based on the semantic descriptions of services. In the context of data mining, this dynamic variability is desirable, since different algorithms can be used for each one of the five activities presented in the standard flow: selection, pre-processing, transformation, data mining, and post-processing.

4.2. Using semantic web services for achieving reuse and automation

There are two key aspects considered by the data mining sub-framework presented in this article: first, it is desired to reuse existing mining techniques and tools; and second it is also necessary to ensure the automatic discovery, composition and invocation of services by using the semantic web services technology.

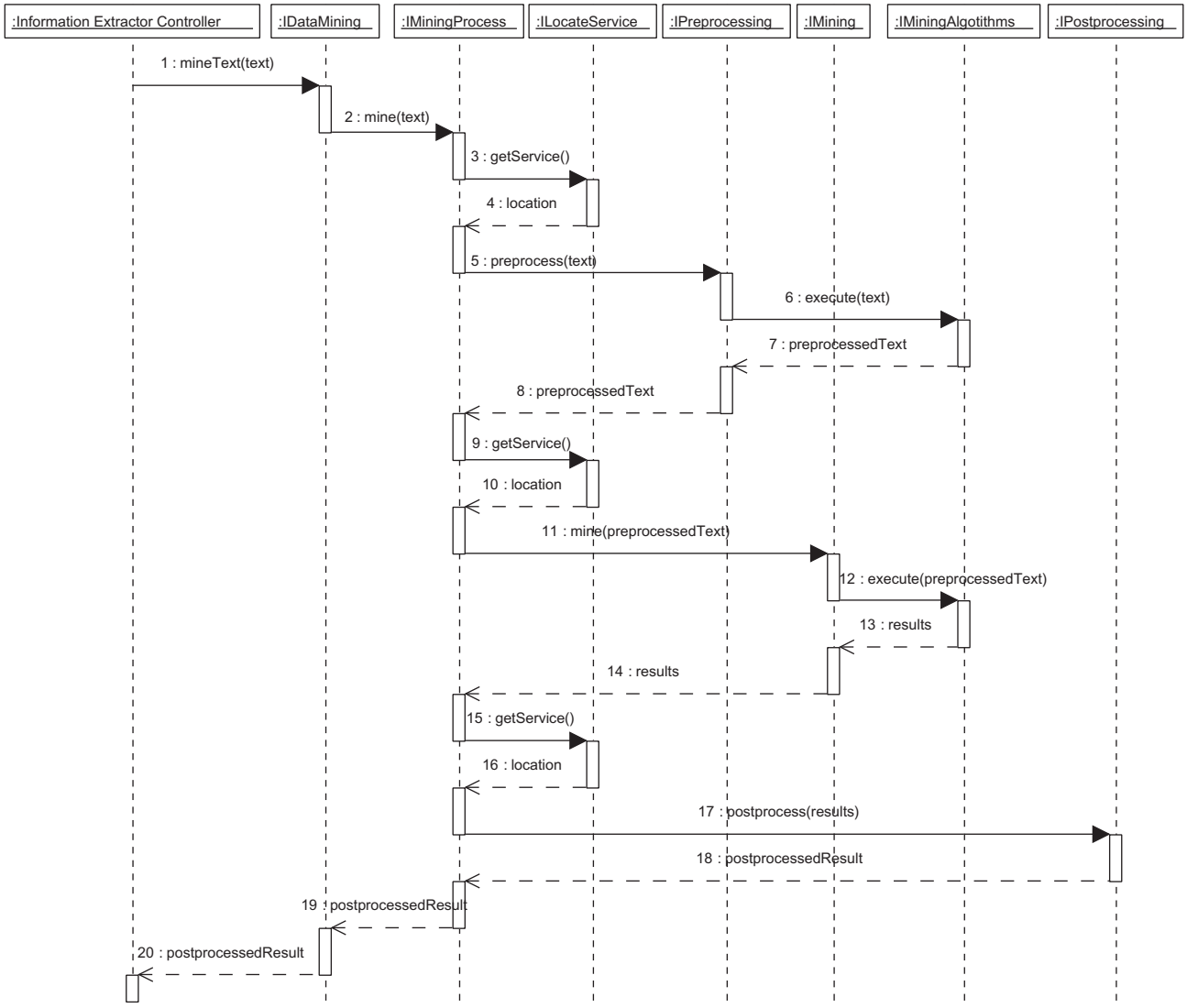

Fig. 22. Sequence diagram of the data mining component of information extractor.

An overview of the structural parts of the data mining sub-framework is presented in Fig. 23, in terms of its main components. The architecture consists of four layers: tools, web services, semantic web services, ontology, and service manager. The role of each layer is described in the following.

4.2.1. Tools

This layer is composed of several mining tools, which provide the mining and preprocessing algorithms that will be used by upper layers to realize data mining process. Each tool has to have a semantic description in OWL[2] so that their functionalities could be interpreted and used by upper layers. It is important to notice that the architecture of the proposed framework allows the addition of new tools (e.g. DBMiner, Clementine). In the context of this article, the chosen tool was Weka (Witten & Frank, 2005) due to its wide use in scientific community, industry, its complete documentation and its easiness of use.

Weka is a collection of machine learning algorithms for common problems in data mining like regression, classification, association rules and attribute selection.

[2] http://www.w3.org/2004/OWL/.

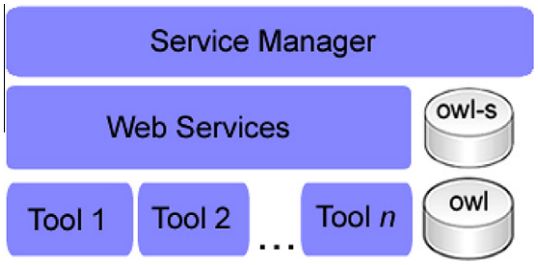

Fig. 23. Data mining sub-framework's architecture.

4.2.2. Web services

This architectural layer supplies an abstraction of the requirements implemented in the data mining tools. This abstraction is achieved through the encapsulation of such requirements in web services. Through this layer, other applications can use mining algorithms by executing the respective wrapping services, with no need of implementing the algorithm itself. As a consequence, such applications are less complex and require less time to be developed.

There are two kinds of Web Services available in this layer: (i) Mining Web Services, which either provide the implementation of the algorithms responsible for executing the mining tasks, for

Fig. 24. Ontology for tasks and techniques.

instance, Apriori (Agrawal & Srikant, 1994) or ID3 (Quinlan, 1986), or encapsulate existing tools that actually implement them; and (ii) Support Web Services, which contain several general-purpose activities, which are necessary to the supplied mining algorithms such as: conversions, web search, noise reduction, inconsistent data detection and others.

4.2.3. Semantic web services

This layer is responsible for mapping the web services available in lower layer, adding semantic to it in order to reach the benefits offered by semantic web services. For illustrating a scenario, suppose the need to obtain association rules from a given dataset. This task is performed in three steps: (i) data preprocessing; (ii) conversion to ARFF format (format accepted by Weka); and (iii) the execution of any association algorithm (e.g. apriori). Using traditional Web Services in each of these steps would require human intervention. Semantically describing Web Services enables configuring software agents to be able to find, compose and invoke these services automatically and transparently to the user. The authors of this paper used OWL-S for semantic description of services. OWL-S has been widely used by the scientific community and is a recommendation of W3C.[3]

4.2.4. Ontology

This layer contains the ontologies which provide a formal knowledge representation. An ontology that semantically describes conceptual and operational aspects related to data mining can be reused by other ontologies that describe data mining and preprocessing tools.

In order to illustrate its usage, Fig. 24 presents an ontology that describes some tasks and techniques for data mining, besides specifying the input and output parameters for each represented learning algorithm.

Four ontology classes are presented in Fig. 24: (i) DataMining, which is a generic class that represents the data mining area (sub-area of Artificial Intelligence); (ii) DataMiningTasks, which represents the data mining tasks such as association, clusterization, classification and other techniques that can also be modeled; (iii) DataMiningTechniques, which represents all the techniques that implement one of the tasks represented in DataMiningTasks class. Each class of DataMiningTask is related with one or more techniques of DataMiningTechniques. Some examples of techniques are: Apriori, *K*-means (Xu & Wunsch, 2005) and C4.5 (Quinlan, 1996); (iv) Parameters, which represents the input and output parameters of the learning algorithms represented in DataMiningTechniques class.

4.2.5. Service manager

This layer has the responsibility is to make the potential benefits of SWS real. In other words, it consists in a Web Service that is in charge of discovering, composing and invoking the available semantic web services. This layer is the boundary between the data mining sub-framework and the client application, which abstracts away the entire process of service discovery, composition and invocation that is done automatically by the Service Manager.

[3] *World Wide Web Consortium* – http://www.w3c.org.

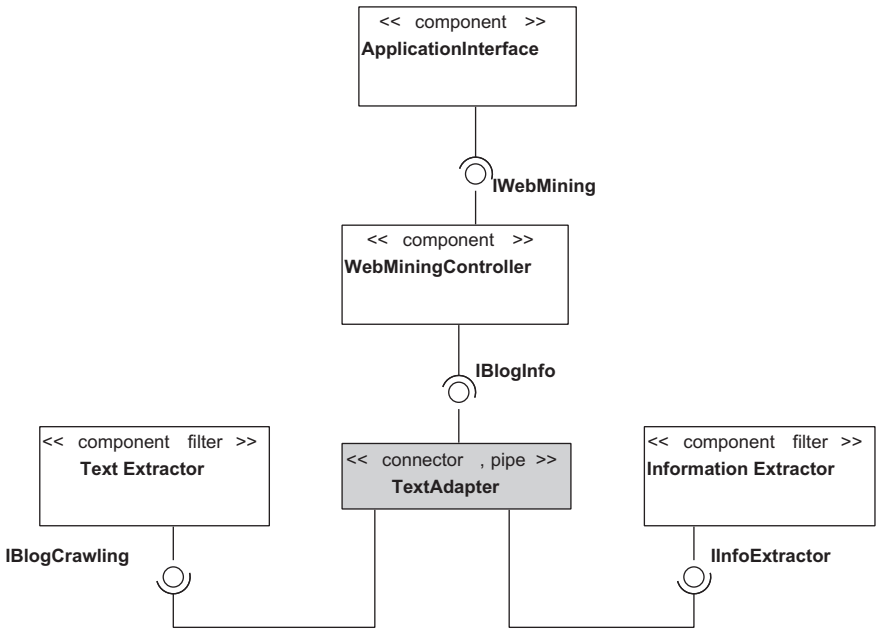

Fig. 25. Software architecture of the case study application.

Table 1
Annotated Blogs.

–	Avatar	The hurt locker
Good comments	30	20
Bad comments	10	20

Thus, the client only needs to know the input and output parameters of the desired service.

The operation of the Service Manager is based on the matchmaking algorithm (Calado, Barros, & Bittencourt, 2009), which is able to discover a service, or a composition of services, which best fits the service the system intends to execute, based on the parameters provided as input.

5. Related work

This section presents some work related to the main contribution of this paper: the data mining sub-framework presented in Section 4. First, existing service-oriented frameworks for data mining are discussed and compared with the proposed approach based on semantic web services (SWS). Then, a general-purpose framework which uses SWS is also considered. To the best of our knowledge, there is no other data mining framework that uses SWS.

Tsai and Tsai (2005) propose a dynamic web service based data mining process system, in which each mining activity is seen as a web service. So, from the user requirements, the available web services are composed together by using a Business Process Execution Language for Web Service (Juric, 2006) (BPEL4WS), for constructing the desired mining process. In this approach, the mining result, which is defined in a Predictive Model Markup Language (PMML), is provided and the analysis service is invoked in order to process them.

Chen, Zhou, and Zhang (2009) propose a web data mining model based on Web services, which consists on a refinement of the Tsai's approach presented already. This work presents a similar architecture and consequently a similar scenario for executing data mining. However, it focus on mining Web content.

The disadvantage of these approaches regards the discovery process and composition of web services, since the developer has to specify a configuration file in which services are explicitly referenced, in order to define the order they will be invoked. This limitation hinders maintainability and dinamicity, since the dependence is defined based on specific services and it is more difficult to add new services, specialy at runtime.

The data mining sub-framework proposed in this article is presented as a solution for the issues previously mentioned, since it uses semantic services to conduct mining, which allows the service manager to perform the discovery process, composition and invocation in an automatic and dynamic way, based on user requirements.

Dietze, Gugliotta, and Domingue (2007) propose a framework based on semantic services for the purpose of providing personalized content to students based on learning objectives defined. For this, learning objects were encapsulated in web services and semantically described using Web Service Modeling Ontology (WSMO), thus allowing the process of composition and invocation to be performed automatically using the Internet Reasoning Service (IRS-III). Thus, based on the goals you want to achieve the most appropriate functionality is selected and invoked, allowing a highly dynamic adaptation to different needs of students.

The data mining sub-framework proposed in this article uses an approach similar to Dietze et al. (2007), but with different goal: perform data mining with a general purpose, which not exclude the education domain. Since the sub-framework allows the definition of variability associated to both the business process and the services for executing each of its activity, its instantiation in different domains is considered a simple task. In the context of the education domain, the sub-framework has been already extended with specialized post-processing services for recommending personalized content to students (Marinho et al., 2010).

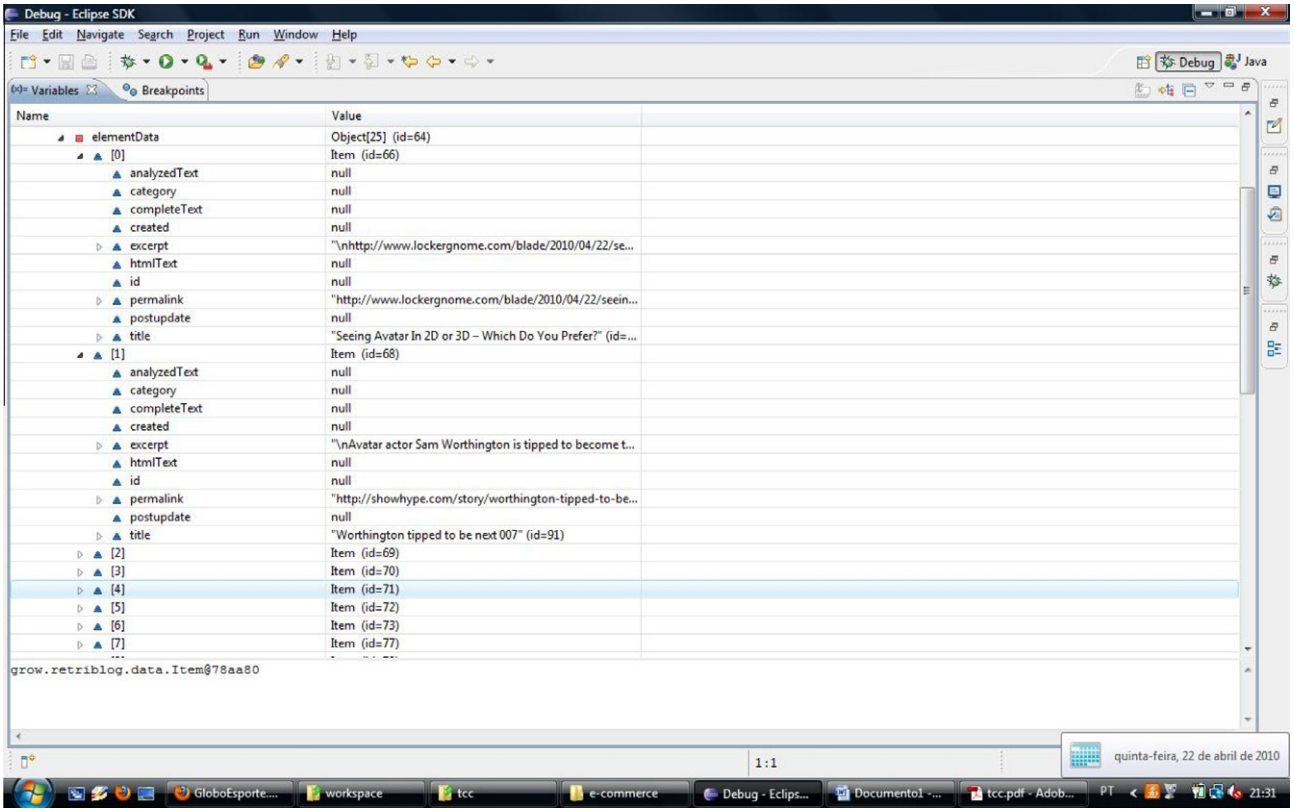

Fig. 26. Result of the obtained Technorati attributes saved in MySQL database.

6. Case study

This case study describes the use of the proposed framework through the implementation of an application to product sentiment analyses. Broadly speaking, the application follows the execution of the sequence diagrams of the text extractor component and also the sequence diagram of the information extractor component.

Indeed, the Fig. 25 describes the implementation of the case study based on the instantiation of the proposed framework. Each component of the software architecture is described as follows.

- ApplicationInterface: this component is responsible for communicating the user with the case study application based on a Web Interface.
- WebMiningController: this component is responsible for providing the communication between the ApplicationInterface component with the framework itself. In other words, the configuration of the framework is provided in this component, describing the whole instances to be used.
- TextExtract: the text extractor has several activities to be performed in order to provide the extracted text. The whole execution is provided on the sequence diagram of the text extractor component.
- InformationExtractor: the execution of this component occurs in the same way of the TextExtractComponent. The whole performation is provided in the sequence diagram of the information extractor component.
- TextAdapter: finally, this component is responsible for providing the communication between the TextExtractor and InformationExtractor component. In other words, this component represents a conector between the core components of the framework.

After the specification of the case study software architecture, it is necessary to follow some steps in order to ensure the right execution of the case study.

6.1. Step 1 – Blog Annotation

This step is responsible for providing the blogs annotated in order to use them to train the mine algorithm. In this case study, 80 blogs were annotated on which half of them are about *avatar* movie and the other part about *the hurt locker* movie, as described on Table 1.

6.2. Step 2 – Configuration of the application

In *Application* class, Figure, it needs to accomplish the following operations:

- *Initialize the numberOfTags variable*: Number of tags that will be used in the application. An integer type.
- *Initialize the tags variable*: Array of *strings* with the tags which whises to extract;
- *Initialize the application Language variable*: Language which whishes to work. At first, it will just be used blogs with the same.
- *Initialize the preprocessing variable*: Preprocessing type that whishes to apply in the text.
- *Initialize the persistence variable*: Persistence type that will be used.

```
public class TechnoratiApplication extends
    Application{

    private int numberOfTags;
    private ArrayList<String> tags;
    private String language;
    private Preprocessing preprocessing;
    private Persistence persistence;
    private Indexing indexing;
    private TextExtraction textExtraction;

    public TechnoratiApplication (){

    TechnoratiApplication
    numberOfTags = 5;
   tags = new ArrayList<String>()
    tags.add("education");
    tags.add("economy");
    tags.add("entertainment");
    tags.add("sports");
    tags.add("movies");

    applicationLanguage = "en";

    preprocessing = preprocessingFactory.
        getEnglishStemming();

    persistence =  persistenceFactory.
        getMySqlPersistence();

    indexing = indexingFactory.getLuceneIndexing();

    textExtraction = textExtractionFactory.
        getSummaryStrategy();

    }

    public boolean checksLanguage(String
        blogLanguage){
 return blogLanguage.equals(this.getLanguage());
    }

    public String preprocesText(String text){
 try {
     return this.getPreprocessing().analyzeString(
         text);
 } catch (IOException e) {
     e.printStackTrace();
     return null;
 }
    }
}
```

Fig. 27. Implementation of class *Application*.

- *Initialize the indexing variable*: Indexing service that will be applied.
- *Initialize the textExtraction variable*: Mechanism of text extraction, from the HTML page, that will be used.
- *Implement the checksLanguage method*: Leaves the *framework* language independent, it is not responsability of the system knows which language will be used, but it is responsability of the application that uses it.
- *Implement the preprocesText method*: The *framework* provides preprocessing services, but it is the application that uses them when necessary.

The Fig. 27 shows the implementation of the *Application* class. In the code we can note that it was created the required attributes and methods to generate the application.

The above code shows the instance of the decision model that is being used. It is in this part, that the user defines which *hot spots* will be used.

```
public ArrayList<Item> tagSearch(String tag) {
    HttpUtils httpUtils = new  HttpUtils();
    ArrayList<Item> technoratiItemList =
    new ArrayList<Item>();

    String page = httpUtils.getPage("http://
        technorati.com"
    + "/tag/" + tag);

    int firstIndexPermalink, lastIndexPermalink,
    firstIndexNamePost, lastIndexNamePost,
    firstIndexExcerpt, lastIndexPermaExcerpt = 0;
    String permalink, namePost, excerpt;
    int links = 0;

    while (links < 20){
        Item item = new Item();
        firstIndexPermalink = page.indexOf("<h3><a
        class=\"offsite\"", lastIndexPermaExcerpt) +
            29;
        lastIndexPermalink = page.indexOf("\">",
        firstIndexPermalink);

        permalink = page.substring(
            firstIndexPermalink,
        lastIndexPermalink);

        firstIndexNamePost = lastIndexPermalink + 2;
        lastIndexNamePost = page.indexOf("</a>",
        firstIndexNamePost);

        namePost = page.substring(firstIndexNamePost
            ,
        lastIndexNamePost);

        firstIndexExcerpt = page.indexOf("<br />",
        lastIndexNamePost + 16) + 6;
        lastIndexPermaExcerpt = page.indexOf("<div
            >",
        firstIndexExcerpt);

        excerpt = page.substring(firstIndexExcerpt,
        lastIndexPermaExcerpt);

        item.setTitle(namePost);
        item.setPermalink(permalink);
        item.setExcerpt(excerpt);
        technoratiItemList.add(item);
        links++;
    }

    return technoratiItemList;
}
      }
```

Fig. 28. Implementation of class *TagParser*.

```
do
    foundBegin <- compare(summary, HTMLBlog)
    if (foundBegin = true)
     startIndex <- index(summary, HTMLBlog)
     foundEnd <- compare(PossibleEndsList, HTMLBlog,
     startIndex)
     if (foundEnd = true)
   endIndex <- index(PossibleEndsList, HTMLBlog,
   startIndex)
   return textBetween(startIndex, endIndex)
     else
   return "not found"
   else
     summary <- divideText(10)

 while (lenght(summary) > 0)
```

Fig. 29. Pseudo code – *SummaryStrategy*.

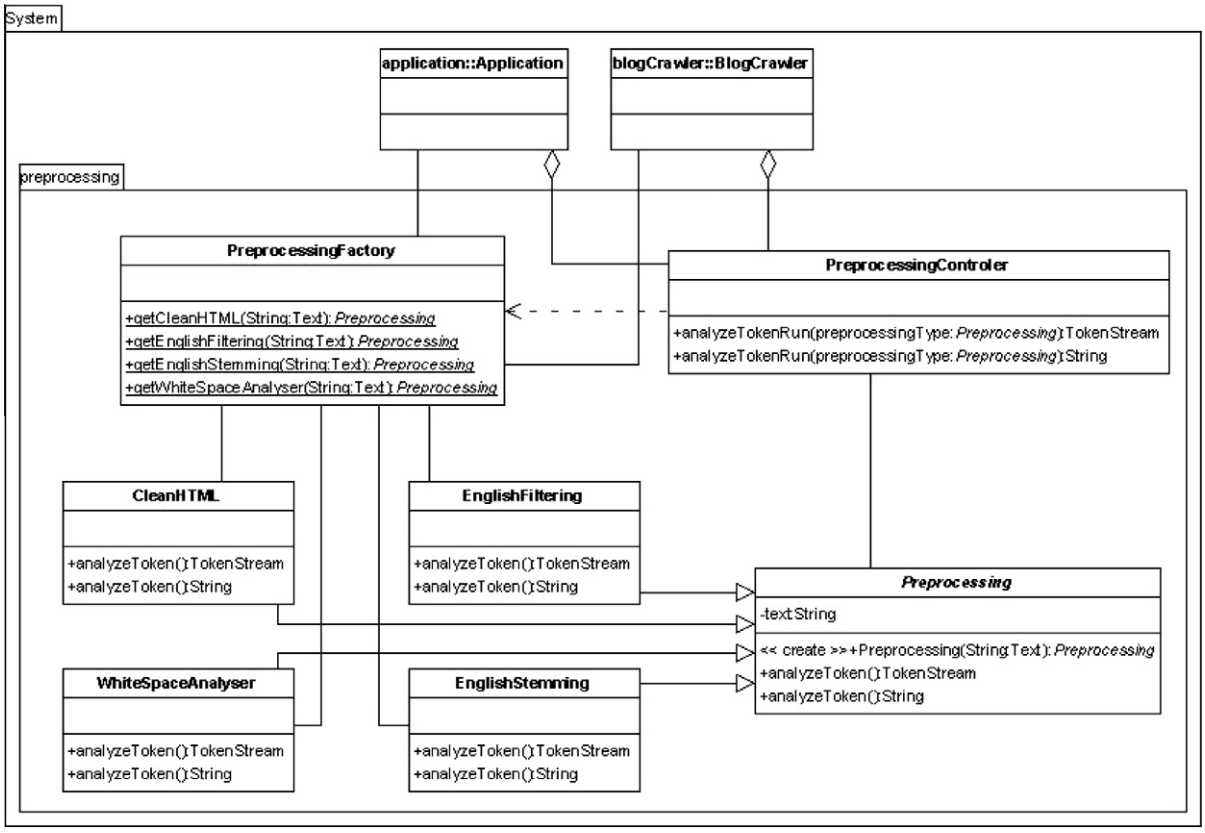

Fig. 30. Diagrama de classes de implementa – *Preprocessing*.

6.3. Step 3 – Fetch blog pages from the Blogosphere

Finally, the implementation of the *TagParser* class. Following the explanation of the *TagParser* code (see Fig. 28) and also the result (see Fig. 26):

1. Initialization of basic variables.
2. Retrieve of the *Tecnhorati* page that contains the blogs of a certain tag.
3. Initialization of auxiliary variables.
4. Parser to obtain the attributes of blogs contained on the page of the *Tecnhorati*.
5. Creation of the item attributes to be added to the return array.

6.4. Step 4 – Extract blog texts from blog pages

A problem found during the implementation of the *framework* was that no search engine, for blog, provides services that return only the content of the post. Furthermore, the blogs do not have an universal structure, that is, each blog has his own structure of HTML tags, difficulting the text extraction using HTML specialized APIs. For this reasons, the work also proposes an algorithm to extract the text of the blog posts from its HTML code.

As input to the algorithm are needed the link and the beginning of the post. A problem found is that the summary which the tools provide are usually returned in string format, that is, lose all the tags that are in it, for example, HTML pages have specific tags to leave the characters in blog, in this case, this tags are lost, so if we get the String with the summary and compare with the HTML string we will not probably found equality between them. Therefore, besides using the summary, it is also divided and compared again, until the text begin is identified. On the other hand, although it is not be structured, there are some tags that are responsible for finish a block of a HTML page, with the tag , they will be used to define the end of the text. Next it will be showed an pseudocode for best understanding (see Fig. 29), the used functions and variables are listed as follows:

- foundBegin: Determine if it found the beginning of the text (boolean variable).
- summary: Summary of the post that wishes to extract (text variable).
- HTMLBlog: Full HTML of the blog (text variable);
- beginIndex: Beginning index of the HTML that was found the text (numeric variable).
- foundEnd: Determine if it found the end of the text (boolean variable).
- PossibleEndsList: List of the possible end for the text, predefined (text list).
- endIndex: End index of the HTML that was found the text (numeric variable).

- compare (*x*,*y*): Compare two texts and returns if exists the text *x* in *y*.
- compare (*x*,*y*,*z*): Compare two texts and returns if exists the text *x* in *y*, from *z*.
- index (*x*,*y*): returns the index it found the text.
- index (*x*,*y*,*z*): Returns the index it found the text, from *z*.
- divideText (*x*): Remove *x* characters from the end of the text.
- lenght (*x*): Returns the lenght of the text variable *x*.

In order to provide one example of this step, the following blog (http://www.afterellen.com/blog/d-word/the-weekly-geek-avatar-gets-an-undersea-sequel) was fetched through the instantiated application and the execution of this step provided the result described below.

```
<p>Do we have any graphic design goddesses in the audience? Well,
you don t have to be to appreciate this, the coolest <a href="..."
target="_Blank">Photoshop tutorial</a> I ve ever seen on the internet.
In fact, sci-fi and video game fans should go give it a look as well,
as it takes budding designers step-by-step through the creation of an
amazing-looking sci-fi computer game poster."\nHow can you not love
this description:\nAs for sci-fi news of the sit down and watch variety,
there s a whole slew of <em><a href="http://www.afterellen.com/taxonomy/term/2105">
Avatar</a></em> news on the docket. The DVD and Blu-Ray release is actually today
&mdash; so go get your overblown space epic if you re so inclined. Didn t get enough?
Well, according to a news item at the <a href="..." target="_Blank"> Underwire blog</a>,
the flick will be re-released in theaters this August with six minutes of added footage.
\nThe same post has some tasty details on the film s sequel. Guess what? It s going to be
set under the sea, as Cameron has stated that the second installment will be focusing on
the ocean on Pandora. \nCameron s fascination with oceans stretches back more than two
decades. He developed much of the 3-D technology used in <em>Avatar</eM> while working
on his post-<eM> Titanic</eM> documentary, <Em> Ghosts of the Abyss</eM>, and took
moviegoers beneath the sea in the 1989 epic <EM>The Abyss</eM>.\nThat will be truly
breathtaking &mdash; and hard to pull off. The world of Pandora is lush and alien and
incredible looking as is, so setting it all under digital water (or perhaps not water
at all) with a host of alien sea life sounds appropriately challenging for Mr. King
of the World.\nWhat do you think readers? Have you had enough of the blue cat people,
or are you itching for more <EM>Avatar</em>?\n
```

6.5. Step 5 – Blog HTML preprocessing

The preprocessing has services that analyses the text of the blog to remove information with little relevance. There are four types of preprocessing services already implemented, as described in Fig. 30. This step performs the CleanHTML, which is responsible for clearing all HTMLs in the text.

In order to show the use of this step, the example below shows the HTML preprocessing of the page described in the Step 4.

```
Do we have any graphic design goddesses in the audience? Well, you don t have to
be to appreciate this, the coolest Photoshop tutorial I ve ever seen on the
internet. In fact, sci-fi and video game fans should go give it a look as well,
as it takes budding designers step-by-step through the creation of an amazing-looking
sci-fi computer game poster."\nHow can you not love this description:\nAs for sci-fi
news of the sit down and watch variety, there s a whole slew of Avatar news on the
docket.
The DVD and Blu-Ray release is actually today so go get your overblown space epic if
you re so inclined. Didn t get enough? Well, according to a news item at the
Underwire blog, the flick will be re-released in theaters this August with six
minutes of added footage. \nThe same post has some tasty details on the film s
sequel. Guess what? It s going to be set under the sea, as Cameron has stated that
the second installment will be focusing on the ocean on Pandora. \nCameron s
fascination with oceans stretches back more than two decades. He developed much
of the 3-D technology used in Avatar while working on his post-Titanic documentary,
Ghosts of the Abyss, and took moviegoers beneath the sea in the 1989 epic The Abyss.
\nThat will be truly breathtaking and hard to pull off. The world of Pandora is
lush and alien and incredible looking as is, so setting it all under digital water
(or perhaps not water at all) with a host of alien sea life sounds appropriately
challenging for Mr. King of the World.\nWhat do you think readers? Have you had
enough of the blue cat people, or are you itching for more Avatar?\n
```

6.6. Step 6 - Blog text preprocessing

This step describes the preprocessing of the blog text through the use of the EnglishFiltering algorithm, which filtering technique (removal of stop words) for English.[4]

The result of the page below is described as follows:

```
do we have ani graphic design goddess audienc well you don t have appreci
coolest photoshop tutori i ve ever seen internet fact sci fi video game fan
should go give look well take bud design step step through creation amaz look
sci fi comput game poster how can you love descript sci fi news sit down watch
varieti s whole slew avatar news docket dvd blu ray releas actual today so go
get your overblown space epic you re so inclin didn t get enough well accord news
item underwir blog flick re releas theater august six minut ad footag same post
has some tasti detail film s sequel guess what s go set under sea cameron has
state second instal focus ocean pandora cameron s fascin ocean stretch back more
than two decad he develop much 3-d technolog use avatar while work his post titan
documentari ghost abyss took moviego beneath sea 1989 epic abyss truli breathtak
hard pull off world pandora lush alien incred look so set all under digit water
perhap water all host alien sea life sound appropri challeng mr king world what
do you think reader have you had enough blue cat peopl you itch more avatar
```

[4] The others preprocessing steps were not necessary to use.

6.7. Step 7 – Classification

In this step, we have the correct pair <text, tag> (obtained by the IBlogCrawling) to create the classification model for the sentiment analysis problem using a learning algorithm (provided by the IInfoExtractor).

Sentiment analysis calls for classifying opinions in text into categories like **positive** or **negative** often with an implicit category of **neutral**. In our experiment we used blog posts about Avatar and The Hurt Locker movies, in an amount of 40 posts about each one. These posts were categorized into **good comment** or **bad comment**. The leaning process was started with 60 blog posts as training dataset and 20 blog posts as test dataset.

To accomplish this classification the IInfoExtractor is responsible for find a service, or set of services, that peform this step based in the received data from IBlogCrawling. The IInfoExtractor uses the Algorithm proposed by Calado et al. (2009) to perform the discovery process.

For this, the desired service was specified according to Table 2, described above. This specification is based on Parameters from the OWL-S Profile of the OWL-S Language (Hobbs et al., 2004).

The Input of the desired service is the Data obtained in the previous steps by IBlogCrawling. The output parameter of the service (i.e. the result of the Classification process) is represented by *AnalysisResult*, that contains the information obtained in the classification process. And finally, ServiceCategory represents a kind of classification of the service, i.e. it is a parameter that helps in refining the search process. In this case is used *NaiveBaiesClassification*, which matches classification process that is desired to use. To change the technique of Classification, it is necessary to change the ServiceCategory.

In other words, the IInfoExtractor is responsible for discover and execute a service that has as Input the *BlogData* and Output an *AnalysisResult* and uses a *NaiveBaiesClassification* algorithm to peform the Classification process.

After the service specification, the algorithm proposed in Calado et al. (2009) tries to find the service in repository that best matches the request. But there are no unique service that has the *NaiveBaiesClassification* ServiceCategory has an *ARFF* as Input and a *Model* as Output.

Then, the algorithm needs to find a composition of services that can provide the features described above. As a result of the execution of this algorithm is obtained as a result the composition of services: (i) *Data2ARFF*, (ii) *NaiveBaiesClassification*, and (iii) *SentimentAnalysis*. Parameters of these services are described in Table 3, as can be seen, the invocation of these services sequentially provides the execution of a service consisting of classification, which uses the *NaiveBaiesClassification* technique and has *BlogData* as input, and as output *AnalysisResults*, which fits describing the desired service. The information about the execution of these services are described above, but more informations should be found in Marinho et al. (2010).

The *Data2ARFF* service is a transformation service that is responsible for receive data in diferent formats and put them in *ARFF* format. The *ARFF* format is used in the *NaiveBaiesClassification* process and in some other Classification techniques present in IInfoExtractor. So, the execution of this service creates an *ARFF* structure that contains the *BlogData* and can be used by the next service.

Table 2
Service description.

Descriptions	
Input	BlogData
Output	AnalysisResult
ServiceCategory	NaiveBaiesClassification

Table 3
Service Composition discovered.

Data2ARFF service	
Input	BlogData
Output	ARFF
NaiveBaies Classification service	
Input	ARFF
Output	Model
Sentiment Analysis service	
Input	Model
Output	AnalysisResult

The *NaiveBaiesClassification* service used the data and the parameters received from the previous service for building a classification model used in post-processing. The model created has an accuracy of 85%, i.e. sentiment analyzer created by the IInfoExtractor correctly classified 17 of 20 posts in the test dataset. This process is described in the next subsection.

The post-processing phase is performed by *SentimentAnalysis* service and consists in using the classifier learned to analyze automatically new or other posts about the movies listed in previously. The possibility of the use of the sentiment analysis are many, a classic sentiment application would be tracking what bloggers are saying about these movies.

Even the number of posts being small, the learned model achieved good results. Part of the 15% of error is on posts that are in the implicit category of **neutral**. However, the main objective of this case study was not to optimize the learning process but to show how easy is to build web mining applications using our framework.

6.7.1. Naive Bayes classifiers

Naive Bayes classifiers (Mitchell, 1997) are known as a simple Bayesian classification algorithm. It has been proven very effective for text categorization. Regarding the text categorization problem, a document $d \in D$ corresponds to a data instance, where D denotes the training document set.

The document d can be represented as a bag of words. Each word $w \in d$ comes from a set W of all feature words. Each document d is associated with a class label $c \in C$, where C denotes the class label set.

The Naive Bayes classifiers estimate the conditional probability $P(c|d)$ that represents the probability that a document d belongs to a class c. Using the Bayes rule, we have $P(c|d) \alpha P(c) \cdot P(d|c)$.

The key assumption of Naive Bayes classifiers is that the words in the documents are conditionally independent given the class value, so that $P(c|d) \alpha P(c) \prod_{w \in d} P(w|c)$.

A popular way to estimate $P(w|c)$ is through Laplacian smoothing: $P(w|c) = (1 + n(w,c))/(|W| + n(c))$ where $n(w,c)$ is the number of the word positions that are occupied by w in all training examples whose class value is c. $n(c)$ is the number of word positions whose class value is c. Finally, $|W|$ is the total number of distinct words in the training set.

7. Conclusions and future work

This article has presented a software framework for building blog mining applications in electronic commerce scenario, providing a set of facilities to alleviate the work of application developers. The proposed framework has contributions which differ it from other existing frameworks, specially in the context of the reduction of effort for developing and maintaining mining applications in

e-commerce based on the framework variabilities. The use of semantic web services is considered very important for achieving this benefit, since it improves the system autonomy by allowing automatic discovering, composition and execution of services that have different (but semantically equivalent) contract description. In the context of software engineering, the most important benefit of using semantic web services is the considerable reduction of the coupling between services, which has a positive impact into the system understandability and maintainability. The low-coupling is achieved by using individual and independent service contract definition through ontologies, instead of a single WSDL file. Moreover, web services are also used for encapsulating existing tools and maximize reuse.

The performed validation has shown the approach is useful. Although this validation cannot be considered sufficient to be fully conclusive, once just one experiment was performed, preliminary results are very encouraging. However, as we have concretely evaluated the proposal for only one experiment, the immediate future work is the need for more experiment. Additionally, other algorithms should be experimented in the sentiment analysis application.

A limitation of this work concerns the need of manually annotating the blog content in order to improve the precision of the crawling task. Moreover, in case of blogs that have multiple content (e.g., comment different kinds of sports), Natural Language Processing (NLP) techniques would be necessary (Liu, 2010). Despite these limitations, the proposed approach is very useful in the context of blogs of a single subject, such as those used for promoting movies and other specific products on the Web, as well as other Web medias, such as forums. Moreover, the data mining subframework presented in Sections 3.2 and 4 can be used for mining other kinds of content with specific purposes, such as for content recommendation in the context of e-learning (Marinho et al., 2010). A future, but complex work, is to develop a tool for automatically annotating blogs by using NLP techniques.

References

Agrawal, R., & Srikant, R. (1994). Fast algorithms for mining association rules in large databases. In *VLDB'94: Proceedings of the 20th international conference on very large data bases* (pp. 487–499). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Bass, L., Clements, P. C., & Kazman, R. (2003). *Software architecture in practice* (2nd ed.). Addison-Wesley.

Boley, H., Grosof, B., Battle, S., Bernstein, A., & Gruninger, M. (2005). *Semantic web services framework (SWSF)*. <http://www.w3.org/Submission/2005/07/>. Accessed June 2010.

Buschmann, F., Meunier, R., Sommerlad, P., & Stal, M. (1996). *Pattern-oriented software architecture: A system of patterns*. John Wiley and Sons.

Calado, I., Barros, H., & Bittencourt, I. I. (2009). An approach for semantic web services automatic discovery and composition with similarity metrics. In *SAC'09: Proceedings of the 2009 ACM symposium on applied computing* (pp. 694–695).

Chen, C., Zhou, X., & Zhang, J. (2009). Web data mining system based on web services. In *HIS '09: Proceedings of the 2009 ninth international conference on hybrid intelligent systems* (pp. 216–220). Washington, DC, USA: IEEE Computer Society.

Cho, J., & Tomkins, A. (2007). Guest editors' introduction: Social media and search. *IEEE Internet Computing, 11*(6), 13–15.

Daconta, M. C., Obrst, L. J., & Smith, K. T. (2003). *The semantic web: A guide to the future of XML, web services, and knowledge management*. Indianapolis: Wiley.

Dietze, S., Gugliotta, A., & Domingue, J. (2007). A semantic web service oriented framework for adaptive learning environments. In *ESWC '07: Proceedings of the 4th European conference on the semantic web* (pp. 701–715). Berlin, Heidelberg: Springer-Verlag.

Erl, T. (2005). *Service-oriented architecture: Concepts, technology, and design*. Upper Saddle River, NJ, USA: Prentice Hall, PTR.

Gupta, P., & Johari, K. (2009). Implementation of web crawler. In *2nd International conference on emerging trends in engineering and technology (ICETET)* (pp. 838–843).

Hatcher, E., & Gospodnetic, O. (2004). *Lucene in action (in action series)*. Greenwich, CT, USA: Manning Publications Co.

Hobbs, J., Lassila, O., McDermott, D., McIlraith, S., Narayanan, S., & Paolucci, M., et al. (2004). *OWL-S: Semantic markup for web services*. <http://www.w3.org/Submission/2004/SUBMOWLS-20041122/>. Accessed June 2010.

Hurst, M., & Maykov, A. (2009). Social streams blog crawler. In *IEEE 25th international conference on data engineering, ICDE'09* (pp. 1615–1618).

Jacobson, I., Booch, G., & Rumbaugh, J. (1999). *The unified software development process*. Boston, MA, USA: Addison-Wesley, Longman Publishing Co., Inc.

Juric, M. B. (2006). *Business process execution language for web services BPEL and BPEL4WS* (2nd ed.). Packt Publishing.

Kruchten, P., Henk Obbink, J., & Stafford, J. A. (2006). The past, present, and future for software architecture. *IEEE Software, 23*(2), 22–30.

Larman, C. (1998). *Applying UML and patterns: An introduction to object-oriented analysis and design*. Prentice-Hall.

Liu, B. (2010). Sentiment analysis and subjectivity. In N. Indurkhya & F. J. Damerau (Eds.), *Handbook of natural language processing* (2nd ed.. Boca Raton, FL: CRC Press, Taylor and Francis Group. ISBN: 978-1420085921.

Marinho, T., Costa, E. B., Dermeval, D., Ferreira, R., Braz, L. M., Bittencourt, I. I., et al. (2010). An ontology-based software framework to provide educational data mining. In *SAC'10: Proceedings of the 2010 ACM symposium on applied computing* (pp. 1433–1437). New York, NY, USA: ACM.

Mary, S., & Clements, P. (1997). A field guide to boxology: Preliminary classification of architectural styles for software systems. In *Proceedings of COMPSAC97, first international computer software and applications conference*.

McIlraith, S. A., Son, T. C., & Zeng, H. (2001). Semantic web services. *IEEE Intelligent Systems, 16*, 46–53.

Mitchell, T. (1997). *Machine learning (Mcgraw-Hill International Edit)* (1st ed.). McGraw-Hill Education (ISE Editions). ISBN: 0071154671.

Parnas, D. L. (1972). On the criteria to be used in decomposing systems into modules. *Communications of the ACM, 15*, 1053–1058.

Pressman, R. S. (2001). *Software engineering: A practitioner's approach* (5th ed.). McGraw-Hill.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning, 1*(1), 81–106.

Quinlan, J. R. (1996). Improved use of continuous attributes in c4.5. *Journal of Artificial Intelligence Research, 4*, 77–90.

Shaw, M., & Garlan, D. (1996). *Software architecture: Perspectives on an emerging discipline* (1st ed.). Prentice Hall.

Travers, G., Domingue, J.B., Polleres, A., Bussler, C., & Burdett, D. (2005). *Web service modeling ontology (WSMO) submission*. <http://www.w3.org/Submission/2005/06/>. Accessed June 2010.

Tsai, C.-Y., & Tsai, M.-H. (2005). A dynamic web service based data mining process system. In *CIT'05: Proceedings of the the fifth international conference on computer and information technology* (pp. 1033–1039). Washington, DC, USA: IEEE Computer Society.

Witten, I. H., & Frank, E. (2005). *Data mining: Practical machine learning tools and techniques* (2nd ed.). Morgan Kaufmann.

WSDL-S submission request to W3C. (2005). <http://www.w3.org/Submission/2005/10/>. Accessed June 2010.

Xu, R., & Wunsch, D. II, (2005). Survey of clustering algorithms. *IEEE Transactions on Neural Networks, 16*(3), 645–678.